The Haizea Manual

1.0 Beta 2

09/04/09

http://haizea.cs.uchicago.edu/

Borja Sotomayor
Department of Computer Science
University of Chicago
borja@cs.uchicago.edu

Contents

Preface

This document is The Haizea Manual, the primary source of documentation for the Haizea lease manager (`http://haizea.cs.uchicago.edu/`). Documentation on features that are in development can be found in our development website at PhoenixForge (`http://phoenixforge.cs.uchicago.edu/haizea`).

This manual can be read as a tutorial introduction to Haizea or as a reference guide. To read it as a tutorial, read Chapters 1 to 4 sequentially (if you are already familiar with what Haizea and resource leases are, you can skip directly to Chapter 3). Additionally, Chapter 6 provides both a tutorial-like introduction and a short reference guide on using Haizea with the OpenNebula virtual infrastructure manager (`http://www.opennebula.org/`). The remaining chapters and appendices are not meant to be read sequentially but, rather, consulted as a reference guide when information is needed on a specific option, functionality, etc.

If you need any help when getting started in Haizea, or need additional information on any of the contents of this manual, please don't hesitate to contact us through our mailing list. You can find instructions on how to subscribe on our website (`http://haizea.cs.uchicago.edu/`) in the "Support" page.

Document conventions

The following conventions are observed in this manual:

```
echo 'This shows something you have to type into your console'
```

```
This shows the contents of a file
```

This is a warning. It indicates that you should proceed with caution.

Part I

Fundamental Concepts

1 What is Haizea?

Haizea is an open-source VM-based lease management architecture. Let's break that down, shall we?

Haizea is a resource manager (or, depending on who you ask, a "resource scheduler"): Haizea is a software component that can manage a set of computers (typically a cluster), allowing users to request exclusive use of those resources described in a variety of terms, such as "I need 10 nodes, each with 1 GB of memory, right now" or "I need 4 nodes, each with 2 CPUs and 2GB of memory, from 2pm to 4pm tomorrow".

Haizea uses leases The fundamental resource provisioning abstraction in Haizea is the lease. Intuitively, a lease is some form of contract where one party agrees to provide a set of resources (an apartment, a car, etc.) to another party. When a user wants to request computational resources from Haizea, it does so in the form of a lease. When applied to computational resources, the lease abstraction is a powerful and general construct with a lot of nuances. Leases are described in more detail in Chapter 2

Haizea is VM-based We hold that the best way of implementing resource leases is using virtual machines (VMs). Therefore, Haizea's scheduling algorithms are geared towards managing virtual machines, factoring in all the extra operations (and overhead) involved in managing VMs. The Globus Virtual Workspaces group, where Haizea was originally developed, has an extensive list of publications that argue how using virtual machines for resource leasing is A Good Thing (and also Not A Trivial Thing).

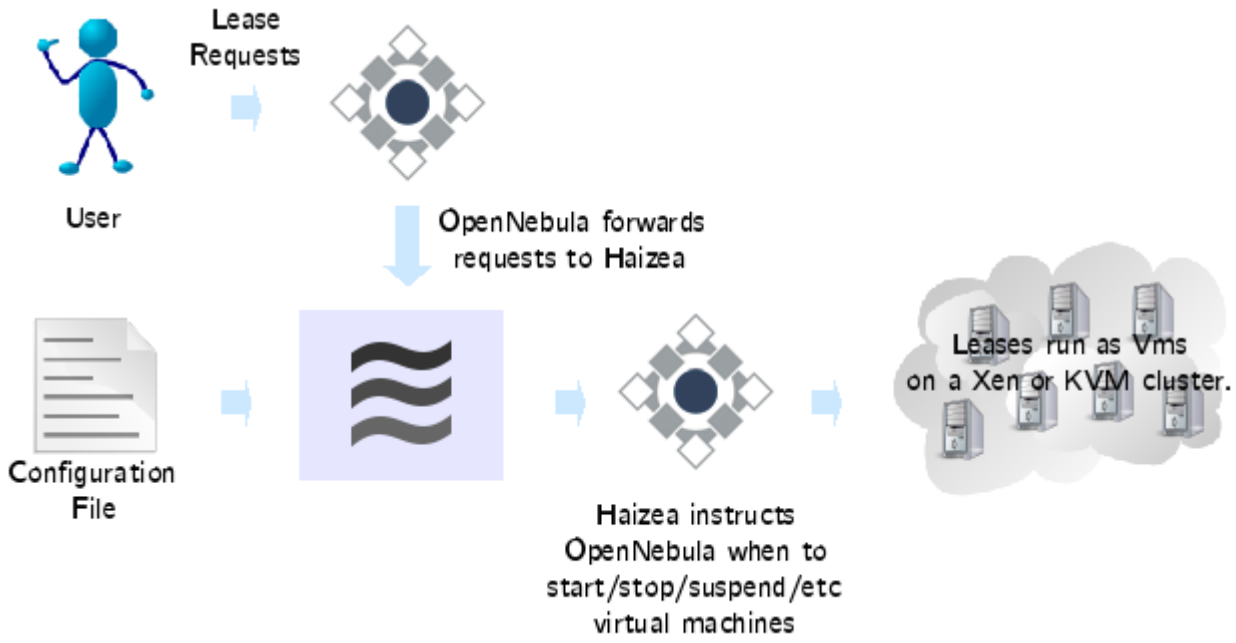
Haizea is open source Haizea is published under the Apache License 2.0, a BSD-like OSI-compatible license.

1.1 What can you do with Haizea?

Haizea is, primarily, a VM resource management component that takes lease requests and makes scheduling decisions based on those requests, but doesn't actually know anything about how to enact those decisions. For example, Haizea may determine at what times a set of VMs representing a lease must start and stop, but it doesn't actually know how to instruct a virtual machine manager (such as Xen, KVM, etc.) to do these actions. Haizea can, however, delegate these enactment actions to an external component using a simple API. Haizea can currently interface with the OpenNebula (`http://www.opennebula.org/`) virtual infrastructure manager to enact its scheduling decisions. Haizea can also simulate enactment actions, which makes it useful for doing scheduling research involving leases or VMs (in fact, the Haizea simulator has been used in a couple of papers).

So, Haizea can be used in three modes: OpenNebula mode, unattended simulation mode, and interactive simulation mode.

1.1.1 OpenNebula mode



Haizea can be used as a drop-in replacement for OpenNebula's scheduling daemon. OpenNebula is a virtual infrastructure manager that enables the dynamic deployment and re-allocation of virtual machines on a pool of physical resources. OpenNebula and Haizea complement each other, since OpenNebula provides all the enactment muscle (OpenNebula can manage Xen and KVM virtual machines on a cluster, with VMWare support to follow shortly) while Haizea provides all the scheduling brains.

Chapter 6 describes how to use Haizea and OpenNebula together.

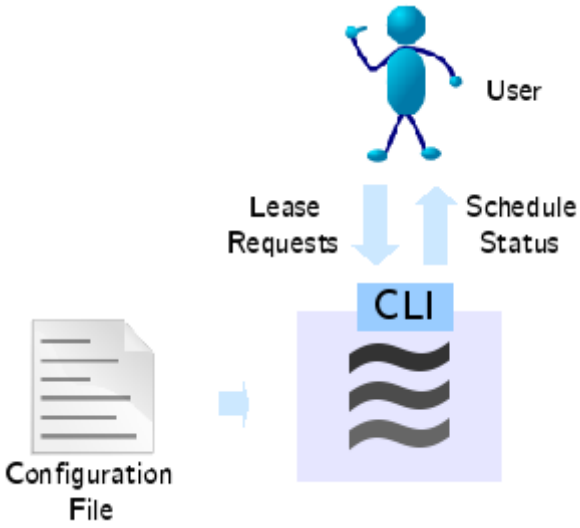
1.1.2 Unattended simulation mode

In this mode, Haizea takes a list of lease requests (specified in a *tracefile*) and a configuration file specifying simulation and scheduling options (such as the characteristics of the hardware

to simulate), and processes them in "simulated time". In other words, the goal of this mode is to obtain the final schedule for a set of leases, without having to wait for all those leases to complete in real time (this makes this mode particularly useful to find out what effect a certain scheduling option could have over a period of weeks or months). In fact, the final result of an unattended simulation is a datafile with raw scheduling data and metrics which can be used to generate reports and graphs.

Chapter 4 provides a quickstart-style introduction to running Haizea in unattended simulation mode, and Chapter 5 explains simulation options in more detail. Analysis of the scheduling data generated by an unattended simulation is covered in Chapter 7

1.1.3 Interactive simulation mode



In this mode, enactment actions are simulated, but Haizea runs in "real time". This means that, instead of having to provide a list of lease requests beforehand, your can use Haizea's command-line interface to request leases interactively and query the status of Haizea's schedule (e.g., to find out the state of lease you've requested). Obviously, this mode is not useful if you want to simulate weeks or months of requests, but it is handy if you want to experiment with leases and track the schedule in a more user-friendly way (since the datafile produced by the unattended simulation is mostly meant for consumption by other programs, e.g., to generate graphs and reports).

Chapter 4, the quickstart-style introduction, also includes instructions on how to run Haizea in interactive simulation mode.

1.2 Haizea architecture

The Haizea architecture (see Figure 1.1) is divided into the following three layers:

The request frontend This is where lease requests arrive. Haizea can currently accept requests from OpenNebula, through a command-line interface, or read them from a tracefile (in SWF format or using the Haizea-specific LWF format).

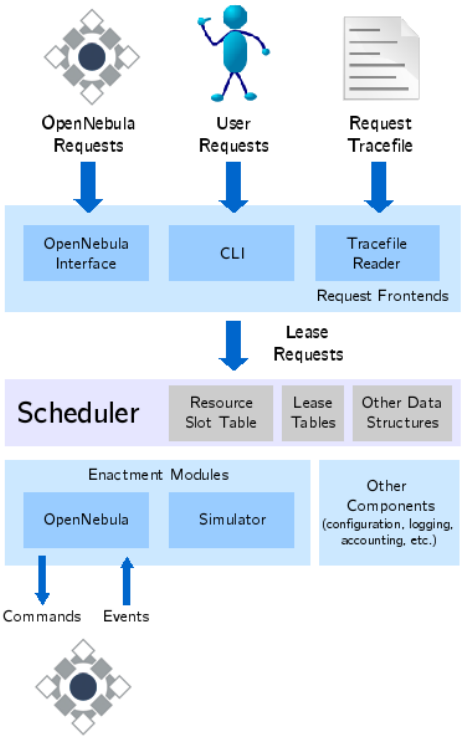


Figure 1.1: The Haizea architecture

The scheduling core This is where the lease requests are processed and scheduled, resulting in enactment actions happening at specific points in time (e.g., "Start VM for lease X in node Y at time T", etc.)

The enactment modules These take care of the "dirty work" of carrying out the enactment actions generated by the scheduler. Haizea can currently send enactment actions to OpenNebula, resulting in Haizea being able to manage Xen and KVM clusters (VMWare support coming soon), or to a simulated cluster.

The Haizea architecture keeps these three layers completely decoupled, which means that adding support for an additional enactment backend only requires writing an enactment module for that backend. The API for enactment modules is still not fully defined, and integration with OpenNebula is currently driving this effort. However, if you'd be interested in using Haizea in another system, please do let us know. We'd be very interested in hearing what your requirements are for the frontend and enactment APIs.

2 Resource leases

Let's say you need computational resources...

Maybe you're a scientist who needs to run some simulations. You have specific hardware requirements, but you're not particularly picky about when the simulations run, and probably won't even notice if they are interrupted at some point, as long as they finish running (correctly) at some point, maybe before a given deadline. A cluster managed by a job scheduler would probably be a good fit for you.

You could also be a software developer who wants to test his or her code on a pristine machine, which you would only need for a relatively short period of time. Plus, every time you use this machine, you'd like it to start up with the exact same pristine software environment. Oh, and you want your machine now. As in *right now*. One option could be to install a virtual machine (VM) manager (such as Xen, VMWare, etc.) to start up these pristine machines as VMs on your own machine. Even better, you could go to a cloud (like Amazon EC2, `http://www.amazon.com/ec2/`, or the Science Clouds, `http://workspace.globus.org/clouds/`) and have those VMs appear automagically somewhere else, so you don't have to worry about setting up the VM manager or having a machine powerful enough to run those VMs.

Or perhaps you're a run-of-the-mill geek who wants his or her own web/mail/DNS/etc server. This server will presumably be running for months or even years with high availability: your server has to be running all the time, with no interruptions. There's a whole slew of hosting providers who can give you a dedicated server or a virtual private server. The latter are typically managed with VM-based datacenter managers.

As you can see, there are a lot of resource provisioning scenarios in nature. However, the solutions that have emerged tend to be specific to a particular scenario, to the exclusion of other ones. For example, while job-based systems are exceptionally good at managing complex batch workloads, they're not too good at provisioning resources at specific times (some job-based systems do offer advance reservations, but they have well-known utilization problems) or at giving users unfettered access to provisioned resources (forcing them, instead, to interact with the resources through the job abstraction).

A lease is a general resource provisioning abstraction that could be used to satisfy a variety of use cases, such as the ones described above. In our work, we've defined a lease as a "negotiated and renegotiable agreement between a resource provider and a resource consumer, where the former agrees to make a set of resource available to the latter, based on a set of lease terms presented by the resource consumer". In our view, the lease terms must include the following dimensions:

Hardware The hardware resources (CPU, memory, etc.) required by the resource consumer.

Software The software environment that must be installed in those resources.

Availability The period during which the hardware and software resources must be available. It is important to note that the availability period can be specified in a variety of ways, like "just get this to me as soon as you can", "I need this from 2pm to 4pm on Mondays, Wednesdays, and Fridays (and, if I don't get exactly this, I will be a very unhappy resource

consumer)", or even "I need four hours sometime before 5pm tomorrow, although if you get the resources to me right now I'll settle for just two hours". A lease-based system must be able to efficiently combine all these different types of availability.

Furthermore, if you don't get any of these dimensions, then you're being shortchanged by your resource lessor. For example, Amazon EC2 is very good at providing exactly the software environment you want, and reasonably good at providing the hardware you want (although you're limited to a few hardware configurations), but not so good at supporting a variety of availability periods.

So, Haizea aims to support resource leasing along these three dimension. For now, Haizea supports three types of availability:

Best-effort lease Resources are provisioned as soon as they are available.

Advance reservation-style leases (or "AR leases") Resources are provisioned during a strictly defined time period (e.g., from 2pm to 4pm).

Immediate leases Resources must be provisioned right now, or not at all.

Although there are many systems (particularly job-based systems) that support the first two types of availability, Haizea differs in that it efficiently schedules heterogeneous workloads (combining best-effort and AR leases), overcoming the utilization problems that tend to occur when using ARs. Haizea does this by using virtual machines to implement leases. Virtual machines also enable Haizea to provide exactly the hardware and software requested by the user. Additionally, Haizea also manages the overhead of preparing a lease, to make sure that any deployment operations (such as transferring a VM disk image) are taken care of before the start of a lease, instead of being deducted from the lessee's allocation.

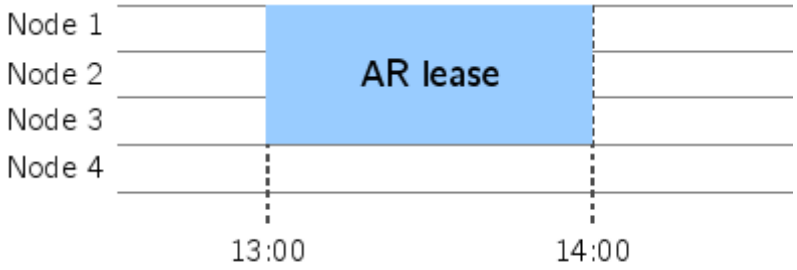
In the future, Haizea will support additional lease types, such as urgent leases, periodic leases, deadline-driven leases, etc.

2.1 Supported types of leases

To better illustrate the types of leases supported in Haizea, let's assume you have a 4-node cluster, and that you want to lease parts of that cluster over time. We'll represent the four nodes over time like this:

2.1.1 "Advance Reservation" lease

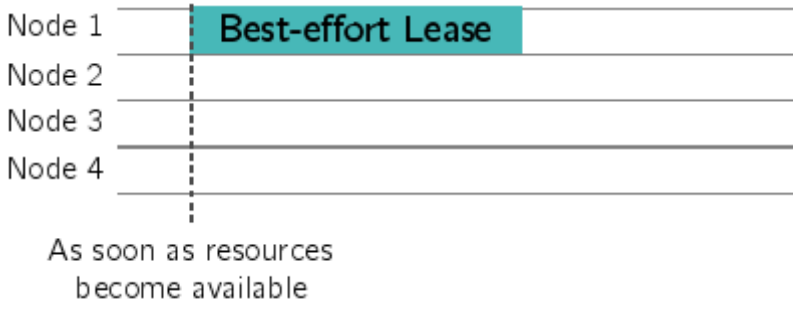
An advance reservation, or AR, lease is a lease that must begin and end at very specific times. For example, the following lease starts at 1pm and ends at 2pm:



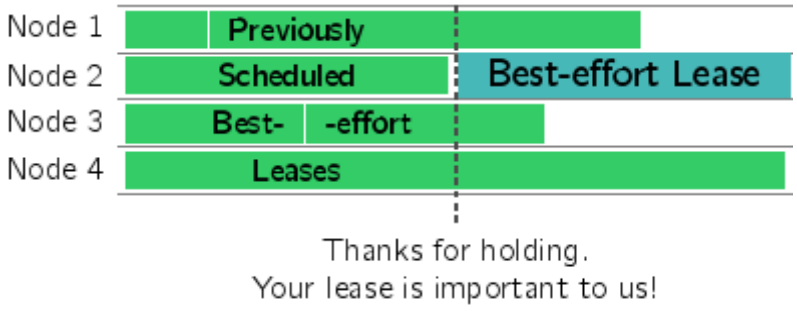
Haizea can schedule this type of lease, which is particularly useful when you need resources at a specific time (for example, to coincide with a lecture, an experiment, etc.)

2.1.2 Preemptible best-effort lease

Sometimes, you know you need resources, but you don't need them at a specific time. In fact, you're perfectly content to wait until there are enough resources available for your lease:



When you request a best-effort lease, your request gets placed in a queue, which is processed in a first-come-first-serve basis (the queue uses backfilling algorithms to improve resource utilization). The downside of this type of lease, of course, is that you may have to wait a while until resources are allocated to your lease:



Furthermore, your lease may be running an unattended program which can be safely paused for a while (since no one is interactively using the lease resources). By requesting a preemptible lease, you allow your resources to be preempted by higher-priority leases, like advance reservations:

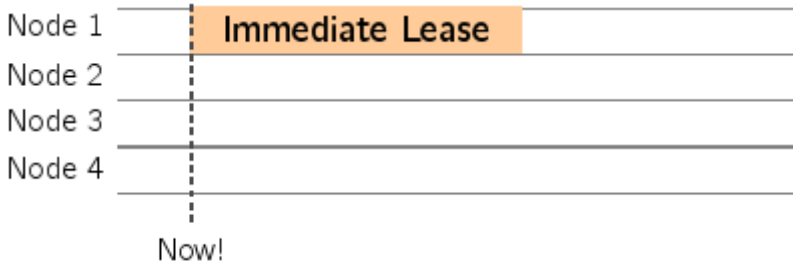
Preemptible best-effort leases are good for running batch jobs, or any non-interactive work. The Haizea paper "Combining Batch Execution and Leasing Using Virtual Machines" showed how using the suspend/resume capability of virtual machines allowed AR and best-effort leases to be scheduled together efficiently, overcoming the utilization problems typically associated with ARs.

2.1.3 Non-preemptible best-effort lease

But what if you're willing to wait for your resources to become available, but don't want them to be preempted? (e.g., if you want to use them interactively). Well, it's as simple as requesting a non-preemptible best-effort lease. Once your request makes it through the queue, and your lease is allocated resources, no one is taking them away.

2.1.4 Immediate lease

In some cases, you may need resources now. As in *right now*:



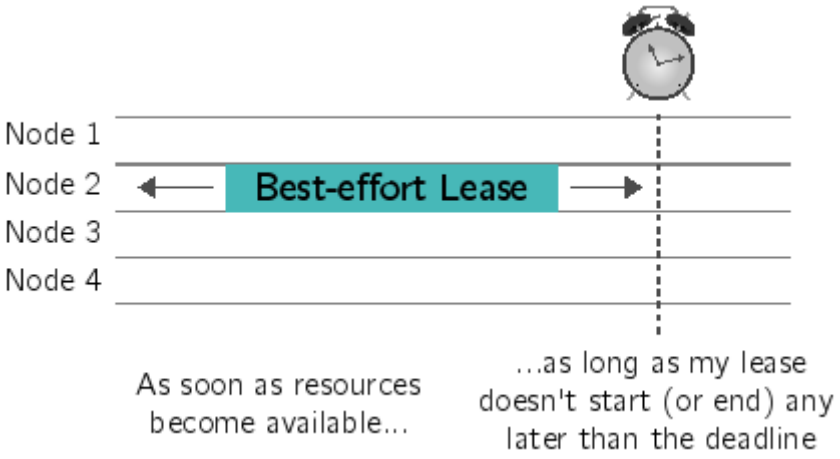
Furthermore, if you can't get them right now, you're just not interested in anything else the resource provider has to offer. You're not going to request resources in the future, and you're certainly not going to be put on a queue. This is essentially the type of lease that many cloud systems offer (although the definition of "right now" varies wildly). Take into account that an immediate lease may still take a while to setup (VM image deployment, etc.). This type of lease in Haizea may evolve in the future into an "urgent lease", where "right now" really does mean "right now".

2.1.5 Coming soon...

In the future, Haizea will support more types of leases, such as best-effort leases with deadlines and leases requiring a non-trivial negotiation before the lease is accepted.

Best-effort with deadlines

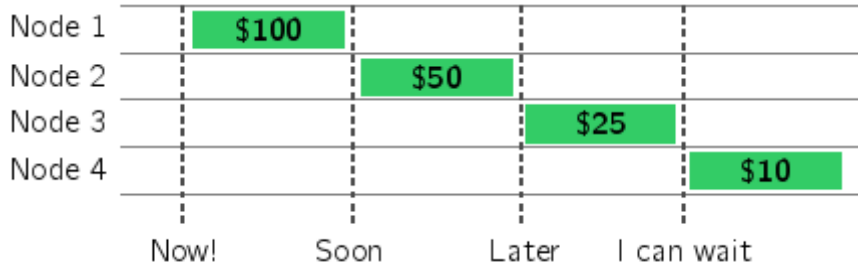
In some cases, when you say "best effort", you really mean "best effort, but be reasonable". Sure, you're willing to wait for your resources, but you may need them before a deadline.



For example, let's say you want a 16-node cluster sometime today to run a test program. You're not particularly picky about when you get the cluster, as long as it happens today and you're given sufficient warning of when your lease will be available. In the future, you will be able to tell Haizea that you have a deadline, and Haizea will either get the resources to you by then, or tell you that the deadline is simply unfeasible.

Negotiated leases

If you've ever entered into any sort of non-computational lease agreement, you know that agreeing on the lease terms rarely involves the lessor instantly being on the same page as you. Rather, it involves a fair amount of haggling. Besides, if your computational needs are flexible, so should your lease manager (c'mon, are you sure you mean "exactly at 2pm"? maybe you meant to say "at some point this afternoon"?). In the future, you will be able to negotiate your leases with Haizea:



So, hey, maybe we can't get you that shiny AR you want at 2pm, but how about I get you twice the resources at an off-peak time? I'll even throw in a discount. And free air conditioning.

Part II

Using Haizea

3 Installing Haizea

Haizea has been tested only on Unix systems, and these installation instructions are given with a Unix system in mind. However, Haizea includes a small amount of platform-specific code, and should run fine on other systems with minimal effort. If there is enough interest, we can produce installers and installation instructions for other platforms.

Installing Haizea can be accomplished in four simple steps:

3.1 Install dependencies

Haizea has a couple of software dependencies. Let's get them out of the way first:

- Python 2.5. (`http://www.python.org/`)
- mxDateTime 3.1.0 (`http://www.egenix.com/products/python/mxBase/mxDateTime/`), part of the eGenix.com mx Base Distribution).
- Optional: Mako Templates for Python 0.2.2 (`http://www.makotemplates.org/`). This package is only necessary if you want to automate running multiple simulation experiments (if this doesn't make any sense, you can skip this prerequisite for now; you will be pointed to this prerequisite again in the documentation when you get to running multiple experiments).
- Optional: Psyco 1.6 (`http://psyco.sourceforge.net/`). This package optimises the execution of Python code, resulting in the simulation code running much faster. You can skip this prerequisite if you are not going to use Haizea to run simulations, or if you are only going to run short simulations.

Note that mxDateTime, Mako, and Psyco are all available as packages (DEB, RPM, etc.) on most Linux distributions. If you don't install any of the optional dependencies, Haizea will still run fine, but some functionality may not be available, as noted above.

3.2 Download Haizea

Go to the download page and download the latest version of Haizea. This will be a tarball called `haizea-XXX.tar.gz`, where XXX will be the version number. For the remainder of the instructions, let's assume that you've saved this file in a directory called `$HAIZEA_INST`.

3.3 Install Haizea

Go into directory `$HAIZEA_INST` and un-tar the installation package:

```
tar xvzf haizea-XXX.tar.gz
```

This will create a directory called `haizea-XXX` in `$HAIZEA_INST`. Go into that directory, and as root, run the following:

```
python setup.py install
```

If you do not have root access, or want to install Haizea in your home directory, run the following:

```
python setup.py install --home=$HOME
```

Note: If you have never installed a Python package in your home directory before, make sure you set the environment variable `PYTHONPATH` appropriately so Python will be aware of the Haizea modules.

```
export PYTHONPATH=$HOME/lib/python
```

After running the setup script, you should see a long list of installation and build messages, ending with the following:

```
creating /usr/share/haizea/traces/multi
copying traces/multi/inj1.lwf -> /usr/share/haizea/traces/multi
copying traces/multi/inj2.lwf -> /usr/share/haizea/traces/multi
copying traces/multi/withprematureend.lwf -> /usr/share/haizea/traces/multi
copying traces/multi/withoutprematureend.lwf -> /usr/share/haizea/traces/multi
running install_egg_info
Writing /usr/lib/python2.5/site-packages/haizea-XXX.egg-info
```

If you see this, installation has been successful!

3.4 Verify installation

Haizea includes some sample configuration files and lease request tracefiles that you can use to test Haizea. If you installed Haizea as root, you can run the following to test your installation:

```
haizea -c /usr/share/haizea/etc/sample_trace.conf
```

This will use a sample configuration file to simulate running the scheduler with no requests, resulting in the following (somewhat anticlimactic) output:

```
[2006-11-25 13:00:00.00] TFILE   Loading tracefile /usr/share/haizea/traces/sample.lwf
[2006-11-25 13:00:00.00] TFILE   Loaded workload with 0 requests ()
[2006-11-25 13:00:00.00] RM      Starting resource manager
[2006-11-25 13:00:00.00] CLOCK   Starting simulated clock
[2006-11-25 13:00:00.00] CLOCK   Simulated clock has stopped
[2006-11-25 13:00:00.00] RM      Stopping resource manager gracefully...
[2006-11-25 13:00:00.00] RM      --- Haizea status summary ---
[2006-11-25 13:00:00.00] RM      Number of leases (not including completed): 0
[2006-11-25 13:00:00.00] RM      Completed leases: 0
[2006-11-25 13:00:00.00] RM      Completed best-effort leases: 0
[2006-11-25 13:00:00.00] RM      Queue size: 0
[2006-11-25 13:00:00.00] RM      Accepted AR leases: 0
[2006-11-25 13:00:00.00] RM      Rejected AR leases: 0
[2006-11-25 13:00:00.00] RM      Accepted IM leases: 0
[2006-11-25 13:00:00.00] RM      Rejected IM leases: 0
[2006-11-25 13:00:00.00] RM      ---- End summary ----
```

Ok, not terribly exciting, but if you see this then the basic machinery is working fine. We will see how to do more elaborate simulations, and how to use Haizea to manage real hardware, in the next chapters.

Note: If you installed Haizea in your home directory, you will have to run the following:

```
haizea -c $HOME/share/haizea/etc/sample_trace.conf
```

Additionally, you will have to modify the tracefile option in the sample configuration so it will point to the sample tracefile located in `$HOME/share/haizea/traces/sample.lwf` (instead of under the `/usr` directory).

4 Quickstart guide

This chapter provides a quick hands-on introduction to using Haizea in simulation mode. Even if you intend to use Haizea in combination with another system, such as OpenNebula, you may still find this guide useful to familiarise yourself with the Haizea configuration file and command-line interface. This chapter assumes that Haizea is installed on your system. If you have arrived at this chapter directly, you may want to read Chapter 1 ("What is Haizea?") first, although you should still be able to follow the instructions in this chapter without reading Chapter 1 first.

4.1 The `haizea` command

The main command in the Haizea system is, unsurprisingly, the `haizea` command. Running this command starts up the Haizea lease manager, which is then ready to receive and schedule lease requests. As described in Chapter 1, Haizea can run in one of three modes: unattended simulated mode, interactive simulated mode, and OpenNebula mode. In this chapter we will focus on the simulation modes, starting with the "unattended" variety. Both simulation modes, and the OpenNebula mode, will be described in more detail in the next chapters.

When running Haizea in unattended simulation mode, the inputs to Haizea are going to be the following:

The Haizea configuration file: A text file containing all the options

A request tracefile: A text file containing a list of lease requests. Since we are using "simulated time", we won't be able to use Haizea interactively (we will be able to do this when we switch to the "real time" mode later in the chapter). Instead, we need to provide all the lease requests beforehand.

Based on the configuration file and the lease requests, the simulator produces a schedule for those leases, which you will be able to follow through logging messages printed by Haizea. At the end of the simulation, Haizea also saves a fair amount of raw data and statistics to disk which can be used to produce reports and graphs (a module to do this for you is in the works). This particular mode, with simulated time, is particularly useful when you want to take a list of request (potentially spanning weeks or months) to see what happens when you tweak the scheduling options (without having to wait weeks or months for the result).

So, let's start by writing a configuration file specifying the simulation options (e.g., the characteristics of the simulated cluster) and the scheduling options.

4.2 The configuration file

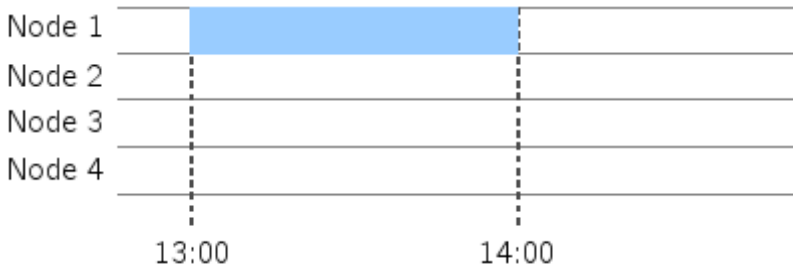
A sample configuration file is provided with Haizea and is located in `/usr/share/haizea/etc/sample_trace.co` (or `$HOME/share/haizea/etc/sample_trace.conf` if you installed Haizea in your home directory). For this guide, you may want to make a copy of this file and use that instead (so you

can preserve the original sample file). If you look at the contents of the file, you will see that it also includes documentation on every option (if you find the documentation gets in the way, there is also a `sample_trace_barebones.conf` file that has the same options, but without any documentation). For now, take a look at the following three options:

```
[simulation]
starttime: 2006-11-25 13:00:00
resources: 4  CPU:100 Memory:1024
```

These options are used to describe the characteristics of our simulated cluster. In particular, we're using a 4-node cluster, each node with 1 CPU, 1024 MB of memory. In this document, we will represent this cluster over time like this:

For example, the following figure shows a lease scheduled on Node 1 from 13:00 to 14:00:



The `starttime` option is used to specify the time at which the simulated clock should start. As you will see, the configuration file has an abundance of other options. We will cover some of them in this chapter, but a more complete reference can be found in Appendix B.

4.3 The tracefile

As mentioned earlier, the simulator will read trace requests from a tracefile. The location of this tracefile is specified in the configuration file, in the `[tracefile]` section:

```
[tracefile]
tracefile: /usr/share/haizea/traces/sample.lwf
```

The default value is a sample tracefile included with Haizea. If you copy the file to a different location, make sure to update the `tracefile` option accordingly. The format of this file is

LWF (Lease Workload Format), an XML format which is particular to Haizea. For now, don't worry about parsing the trace format in detail; it is fairly human-readable and you can also find details on the LWF format in Appendix C.

```
<lease-workload name="sample">
  <description>
  A simple trace where an AR lease preempts a
  best-effort lease that is already running.
  </description>

  <lease-requests>

  <!-- The lease requests are initially commented out -->

  <!-- First lease request -->
  <!--
  ...
  -->

  <!-- Second lease request -->
  <!--
  ...
  -->

  </lease-requests>
</lease-workload>
```

As you can see, there are two lease requests in the file, but they are initially commented out. We will take a closer look at each of these requests next.

4.4 Running the simulator

Now that we have a configuration file and a tracefile, we can run the simulator. You can run Haizea with the sample configuration file like this:

```
haizea -c /usr/share/haizea/etc/sample_trace.conf
```

Which results in the following output:

```
[2006-11-25 13:00:00.00] RM      Starting resource manager
[2006-11-25 13:00:00.00] TFILE   Loading tracefile /usr/share/haizea/traces/sample.lwf
[2006-11-25 13:00:00.00] TFILE   Loaded workload with 0 requests ()
[2006-11-25 13:00:00.00] CLOCK   Starting simulated clock
[2006-11-25 13:00:00.00] CLOCK   Simulated clock has stopped
[2006-11-25 13:00:00.00] RM      Stopping resource manager gracefully...
[2006-11-25 13:00:00.00] RM      --- Haizea status summary ---
[2006-11-25 13:00:00.00] RM      Number of leases (not including completed): 0
[2006-11-25 13:00:00.00] RM      Completed leases: 0
[2006-11-25 13:00:00.00] RM      Completed best-effort leases: 0
[2006-11-25 13:00:00.00] RM      Queue size: 0
[2006-11-25 13:00:00.00] RM      Accepted AR leases: 0
[2006-11-25 13:00:00.00] RM      Rejected AR leases: 0
[2006-11-25 13:00:00.00] RM      Accepted IM leases: 0
[2006-11-25 13:00:00.00] RM      Rejected IM leases: 0
[2006-11-25 13:00:00.00] RM      ---- End summary ----
```

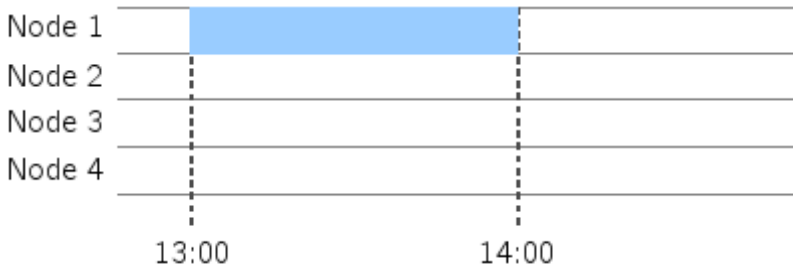
Now that you've seen the tracefile, you can see why the simulator starts up and immediately stops: all the lease requests in the tracefile are commented out, and there's nothing to schedule. Go ahead and uncomment the first lease request, which looks like this:

```
<lease-request arrival="00:00:00">
<lease preemptible="true">
  <nodes>
    <node-set numnodes="1">
      <res type="CPU" amount="100"/>
      <res type="Memory" amount="1024"/>
    </node-set>
  </nodes>
  <start></start>
  <duration time="01:00:00"/>
  <software>
    <disk-image id="foobar.img" size="1024"/>
  </software>
</lease>
</lease-request>
```

This is a request for a best-effort lease (notice how the starting time is left empty, meaning it's up to Haizea to determine the start time), requested at time 00:00:00 (right at the start of the simulation), requiring 1 hour, and only one node. Now run Haizea again. You should now see the following:

```
[2006-11-25 13:00:00.00] RM      Starting resource manager
[2006-11-25 13:00:00.00] TFILE   Loading tracefile /usr/share/haizea/traces/sample.lwf
[2006-11-25 13:00:00.00] TFILE   Loaded workload with 1 requests (1 Best-effort)
[2006-11-25 13:00:00.00] CLOCK   Starting simulated clock
[2006-11-25 13:00:00.00] LSCHED  Lease #1 has been requested.
[2006-11-25 13:00:00.00] LSCHED  Lease #1 has been marked as pending.
[2006-11-25 13:00:00.00] LSCHED  Queued best-effort lease request #1, 1 nodes for 01:00:00.00.
[2006-11-25 13:00:00.00] LSCHED  Next request in the queue is lease 1. Attempting to schedule...
[2006-11-25 13:00:00.00] VMSCHED Lease #1 has been scheduled on nodes [1]
                                 from 2006-11-25 13:00:00.00
                                   to 2006-11-25 14:00:00.00
[2006-11-25 13:00:00.00] VMSCHED Started VMs for lease 1 on nodes [1]
[2006-11-25 14:00:00.00] VMSCHED Stopped VMs for lease 1 on nodes [1]
[2006-11-25 14:00:00.00] VMSCHED Lease 1's VMs have shutdown.
[2006-11-25 14:00:00.00] CLOCK   Simulated clock has stopped
[2006-11-25 14:00:00.00] RM      Stopping resource manager gracefully...
[2006-11-25 14:00:00.00] RM      --- Haizea status summary ---
[2006-11-25 14:00:00.00] RM      Number of leases (not including completed): 0
[2006-11-25 14:00:00.00] RM      Completed leases: 1
[2006-11-25 14:00:00.00] RM      Completed best-effort leases: 1
[2006-11-25 14:00:00.00] RM      Queue size: 0
[2006-11-25 14:00:00.00] RM      Accepted AR leases: 0
[2006-11-25 14:00:00.00] RM      Rejected AR leases: 0
[2006-11-25 14:00:00.00] RM      Accepted IM leases: 0
[2006-11-25 14:00:00.00] RM      Rejected IM leases: 0
[2006-11-25 14:00:00.00] RM      ---- End summary ----
```

The above corresponds to the following schedule:

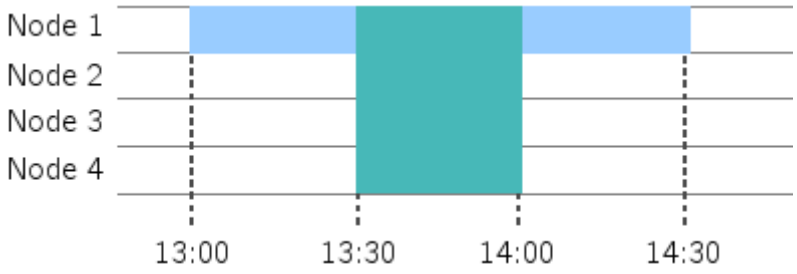


A best-effort request is received at 13:00 and, since the cluster is empty, it is scheduled immediately. Notice how the VMs for the lease start at 13:00 and stop at 14:00. For now, we're assuming that the disk images are predeployed on the physical nodes (we will modify this option in the next section).

Now go ahead and uncomment the second lease request, which looks like this:

```
<lease-request arrival="00:15:00">
<lease preemptible="false">
  <nodes>
    <node-set numnodes="4">
      <res type="CPU" amount="100"/>
      <res type="Memory" amount="1024"/>
    </node-set>
  </nodes>
  <start>
    <exact time="00:30:00"/>
  </start>
  <duration time="00:30:00"/>
  <software>
    <disk-image id="foobar.img" size="1024"/>
  </software>
</lease>
</lease-request>
```

This is a request for an advance reservation lease (notice how there is an exact starting time specified), requesting all four nodes for 30 minutes. So, what would happen if we also added this AR lease? Since it requires all the cluster resources from 13:30 to 14:00, the best-effort lease will be unable to run in that time interval. Since the leases are implemented as VMs, Haizea will still schedule the best-effort lease to start at 13:00, but will suspend it before the AR lease starts, and will resume it once the AR lease has finished. In effect, we want the schedule to look like this:



Uncomment the AR lease request, and run Haizea again. You should now see the following:

```
[2006-11-25 13:00:00.00] RM      Starting resource manager
[2006-11-25 13:00:00.00] TFILE   Loading tracefile /usr/share/haizea/traces/sample.lwf
[2006-11-25 13:00:00.00] TFILE   Loaded workload with 2 requests (1 Best-effort + 1 AR)
[2006-11-25 13:00:00.00] CLOCK   Starting simulated clock
[2006-11-25 13:00:00.00] LSCHED  Lease #1 has been requested.
[2006-11-25 13:00:00.00] LSCHED  Lease #1 has been marked as pending.
[2006-11-25 13:00:00.00] LSCHED  Queued best-effort lease request #1, 1 nodes for 01:00:00.00.
[2006-11-25 13:00:00.00] LSCHED  Next request in the queue is lease 1. Attempting to schedule...
[2006-11-25 13:00:00.00] VMSCHED Lease #1 has been scheduled on nodes [1]
                                 from 2006-11-25 13:00:00.00
                                   to 2006-11-25 14:00:00.00
[2006-11-25 13:00:00.00] VMSCHED Started VMs for lease 1 on nodes [1]

[2006-11-25 13:15:00.00] LSCHED  Lease #2 has been requested.
[2006-11-25 13:15:00.00] LSCHED  Lease #2 has been marked as pending.
[2006-11-25 13:15:00.00] LSCHED  Scheduling AR lease #2, 4 nodes
                                 from 2006-11-25 13:30:00.00
                                   to 2006-11-25 14:00:00.00.
[2006-11-25 13:15:00.00] LSCHED  Must preempt leases [1] to make room for lease #2
[2006-11-25 13:15:00.00] LSCHED  Preempting lease #1...
[2006-11-25 13:15:00.00] LSCHED  ... lease #1 will be suspended
                                     at 2006-11-25 13:30:00.00.
[2006-11-25 13:15:00.00] LSCHED  AR lease #2 has been scheduled.

[2006-11-25 13:29:28.00] VMSCHED Stopped VMs for lease 1 on nodes [1]
[2006-11-25 13:29:28.00] VMSCHED Suspending lease 1...

[2006-11-25 13:30:00.00] VMSCHED Lease 1 suspended.
[2006-11-25 13:30:00.00] VMSCHED Started VMs for lease 2 on nodes [2, 3, 4, 1]
[2006-11-25 13:30:00.00] LSCHED  Next request in the queue is lease 1. Attempting to schedule...
[2006-11-25 13:30:00.00] VMSCHED Lease #1 has been scheduled on nodes [1]
                                 from 2006-11-25 14:00:00.00 (resuming)
                                   to 2006-11-25 14:31:04.00

[2006-11-25 14:00:00.00] VMSCHED Stopped VMs for lease 2 on nodes [2, 3, 4, 1]
[2006-11-25 14:00:00.00] VMSCHED Resuming lease 1...
[2006-11-25 14:00:00.00] VMSCHED Lease 2's VMs have shutdown.

[2006-11-25 14:00:32.00] VMSCHED Resumed lease 1
[2006-11-25 14:00:32.00] VMSCHED Started VMs for lease 1 on nodes [1]

[2006-11-25 14:31:04.00] VMSCHED Stopped VMs for lease 1 on nodes [1]
[2006-11-25 14:31:04.00] VMSCHED Lease 1's VMs have shutdown.
[2006-11-25 14:31:04.00] CLOCK   Simulated clock has stopped
[2006-11-25 14:31:04.00] RM      Stopping resource manager gracefully...
```

Notice how the above corresponds to the previous figure. In particular, notice the following:

- When the AR lease request is received, Haizea looks at the schedule and determines that the only way to schedule the AR lease is to preempt the best-effort lease. However, instead of cancelling that lease, it will just reschedule it so it is suspended right before the AR lease start. Note that Haizea will always try to minimise the number of preemption (in this case, we're forcing the situation for demonstration purposes) by assigning the AR

lease to resources that are available without preempting other leases.

- Shortly before the AR lease starts, the best-effort lease is suspended (the time required to do this is estimated by Haizea based on an option in the configuration file). When the AR lease ends at 14:00, Haizea begins resuming the suspended best-effort lease.

4.5 The scheduling options

Haizea has several scheduling options that control how Haizea selects resources and schedules leases. For example, the above example assumed that leases can be suspended (which they generally always can be when running as virtual machines). What would happen if this were not possible? You can modify the suspension option in the `[scheduling]` section to find out:

```
[scheduling]
...

suspension: none

...
```

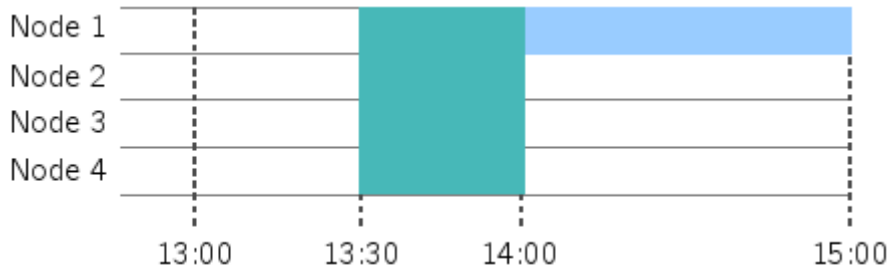
Rerun Haizea. Now, when the AR lease arrives at 13:15, the scheduler will realise it has to preempt the best-effort lease to make room for the AR lease, but will no longer be able to suspend it. The only option is to cancel the best-effort lease and resubmit it to the queue:

```
[2006-11-25 13:15:00.00] LSCHED  Preempting lease #1...
[2006-11-25 13:15:00.00] LSCHED  ... lease #1 has been cancelled and requeued
```

Now, the best-effort lease can only be scheduled after the AR lease, at 14:00:

```
[2006-11-25 13:15:00.00] VMSCHED Lease #1 has been scheduled on nodes [1]
                                 from 2006-11-25 14:00:00.00
                                   to 2006-11-25 15:00:00.00
```

So, the schedule would end up looking like this:



Notice how, although suspending a lease is a disruptive activity which can delay the completion time of a best-effort request, it is still much better than completely cancelling a request and waiting for enough resources to accommodate the entire (uninterrupted) duration of the lease.

Another scheduling option you can modify is whether Haizea should transfer the VM's disk image from an image repository before the lease can start. You can do this by modifying the `lease-deployment` option:

```
[general]
...
lease-preparation: imagetransfer
...
```

If you look at the bottom of the sample configuration file, you will find a section called `[deploy-imagetransfer]` with all the image transfer options.

Rerun Haizea again. You should get a schedule similar to the previous one, but with some extra messages indicating that image transfers are taking place:

```
[2006-11-25 13:00:00.00] DEPLOY  Starting image transfer for lease 1
[2006-11-25 13:01:22.00] DEPLOY  Completed image transfer for lease 1
```

As you can see, the best-effort lease can no longer start right at 13:00, since an image transfer has to take place before the starting time. The same is true of the AR lease, but notice how Haizea schedules the image transfer in such a way that the AR lease can still start at 13:30 as planned (instead of delaying the starting time until 13:31:22).

There are several other options you can modify in the `[scheduling]` section, such as what backfilling algorithm to use, whether to allow lease migration or not, etc. These options are described in the following chapters, and in Appendix B.

4.6 Interactive simulations

Up to this point, Haizea has been scheduling leases in "simulated time". This meant that we provided Haizea with a lease workload beforehand, ran it, and got the results of scheduling that workload much earlier than it would have actually taken to run the leases (e.g., if we requested a 30 minute lease, we didn't have to wait 30 minutes for the lease to complete; Haizea just skipped from the start to the end of the lease). This "fast forward" approach is useful if you want to experiment with different scheduling parameters and workloads. However, you can also run Haizea in simulation and in "real time". To do this, you need to change the `clock` option of the `[simulation]` section:

```
[simulation]
...
clock: real
...
```

If you run Haizea in this mode, it will run a daemon that is ready to accept your requests interactively through a command-line interface, instead of processing a list of requests provided beforehand. You should see the following when running the `haizea` command:

```
Started Haizea daemon with pid NNNN
```

You will then get control of your console back. If you're wondering where all the logging messages are being saved to, they're now being sent to a file. The default logfile is `/var/tmp/haizea.log`. You can take a peek at it like this:

```
tail /var/tmp/haizea.log
```

You will notice messages like this:

```
[2008-09-24 14:14:18.58] CLOCK   Going back to sleep.
                                 Waking up at 2008-09-24 14:15:19.00
                                 to see if something interesting has
                                 happened by then.
```

Since time is not simulated, Haizea doesn't know what the "next time" to skip to will be, so it will simply wake up periodically to see if anything interesting has happened (like a new request). This interval can be changed in the configuration file:

```
[simulation]
...
wakeup-interval: 10
...
```

However, when Haizea plans an event (e.g., leases that have to start or end), it will wake up specifically to handle that event (instead of waiting for the wakeup interval to conclude).

So, let's give Haizea something to do. The `haizea-request-lease` command is used to request leases. For example, the following command is used to request an 1-node AR lease one minute in the future, for ten minutes:

```
haizea-request-lease -t +00:02:00 -d 00:10:00 -n 1 --non-preemptible \
                     -c 1 -m 512 -i foobar.img -z 600
```

Additionally, you can also write a lease request using the XML format seen previous, save it to a file, and have `haizea-request-lease` command parse it:

```
haizea-request-lease -f request.xml
```

You can find more details on this command's parameters by running `haizea-request-lease -h` or taking a look at Appendix A. Once you've submitted the lease, you should see the following:

```
Lease submitted correctly.
Lease ID: 1
```

You can check the status of your submitted lease by looking at the log file or, more conveniently, using this command:

```
haizea-list-leases
```

You should see the following:

```
ID   Type          State      Starting time           Duration      Nodes
1    AR            Scheduled  2009-08-04 11:25:57.00  00:10:00.00   1
```

Note: You may not see your lease right away, since Haizea has to "become aware" of it (which won't happen until it wakes up to check if there are any new requests). Future versions of Haizea will enable it to be notified immediately of incoming requests.

Remember that the lease has been requested one minute into the future, so it will remain in a "Scheduled" state for a couple seconds. If you run `haizea-list-leases` periodically, you should see it pass through a couple other states. If image transfers are still enabled, it will first transition to the "Preparing" state:

```
ID   Type          State      Starting time           Duration      Nodes
1    AR            Preparing  2009-08-04 11:25:57.00  00:10:00.00   1
```

And then to the "Active" state:

```
ID   Type          State      Starting time           Duration      Nodes
1    AR            Active     2009-08-04 11:25:57.00  00:10:00.00   1
```

Now let's request a best-effort lease:

```
haizea-request-lease -t best_effort -d 00:10:00 -n 4 --non-preemptible \
                     -c 1 -m 512 -i foobar.img -z 600
```

The list of leases will now look like this:

```
ID   Type          State      Starting time           Duration      Nodes
1    AR            Active     2009-08-04 11:25:57.00  00:10:00.00   1
2    Best-effort   Scheduled  Unspecified             00:10:00.00   4
```

Note how, for best-effort leases, the starting time is set to "Unspecified", which means this time is not specified by the user, but instead determined on a best-effort basis by the scheduler. Since the lease is in a "Scheduled" state, that means that it has been assigned a starting time (although that information is currently not available through the command-line interface; it can be seen in the Haizea log).

Now try to rerun the `haizea-request-lease` command a couple times (i.e., lets submit a couple more best-effort requests). The scheduler won't be able to schedule them, since they require all the available nodes, and the AR lease is using up one of them. The previous best-effort lease was scheduled because Haizea's default behaviour is to schedule at most one best-effort lease in the future if resources cannot be found right away (this is due to Haizea's use of backfilling algorithms; for now, don't worry if you don't know what they are). Anyway, the list of leases should now look like this:

```
ID   Type  State      Starting time           Duration      Nodes
1    AR            Active     2009-08-04 11:25:57.00  00:10:00.00   1
2    Best-effort   Scheduled  Unspecified             00:10:00.00   4
3    Best-effort   Queued     Unspecified             00:10:00.00   4
4    Best-effort   Queued     Unspecified             00:10:00.00   4
5    Best-effort   Queued     Unspecified             00:10:00.00   4
6    Best-effort   Queued     Unspecified             00:10:00.00   4
```

Notice how the extra best-effort requests have been queued. If you only want to see the contents of the queue, you can use the following command:

```
haizea-show-queue
```

This should show the following:

```
ID   Type          State      Starting time           Duration      Nodes
3    Best-effort   Queued     Unspecified             00:10:00.00   4
4    Best-effort   Queued     Unspecified             00:10:00.00   4
5    Best-effort   Queued     Unspecified             00:10:00.00   4
6    Best-effort   Queued     Unspecified             00:10:00.00   4
```

When you're done, you can shut Haizea down cleanly by running the following:

```
haizea --stop
```

4.7 Other things you can do with Haizea

At this point, we have seen how to run simple simulations with Haizea. However, there is a lot more that Haizea can do:

Run on real hardware First and foremost, almost everything you just saw above in simulation can be done on real hardware. This is accomplished by using Haizea with the OpenNebula virtual infrastructure manager. So, if you have a Xen or KVM cluster, you can just install OpenNebula and Haizea to enable your users to request VM-based leases on your cluster. This is explained in Chapter 6.

Run complex simulations This chapter concerned itself mostly with scheduling two leases on a 4-node cluster during a span of roughly 2 hours. *Boring.* Haizea can handle more complex simulations, and also provides the necessary tools for you to easily run multiple simulations with different profiles. For example, in the Haizea paper "Combining Batch Execution and Leasing Using Virtual Machines" (see the Haizea publication page: `http://haizea.cs.uchicago.edu/pubs.html`) we simulated running 72 30-day workloads in six different configurations, or 36 years of lease scheduling. Running multiple simulations is explained in Section 5.5

Produce reports and graphs The above examples relied on reading the Haizea log messages or peeking into Haizea's schedule using command-line tools. This is ok for a simple simulation, but no fun when you're scheduling thousands of leases. Haizea saves a fair

amount of raw data to disk with scheduling metrics, utilization information, etc. which can be used to generate reports and graphs. We are in the process of producing tools that will allow you to easily analyse that data and create graphs, although some pointers on how to interpret the raw data produced by Haizea are presented in Chapter 7.

5 Running scheduling simulations

This chapter describes how to run Haizea in simulation mode. Since the Quickstart Guide (Chapter 4) already provides a tutorial-like introduction to running simulations, this chapter is meant mostly as a reference guide, and covers the main simulation and scheduling options. However, it does not cover *all* possible options in the configuration file (a description of all options and their valid values can be found in Appendix B). It also refers to scheduling algorithms that are not currently explained in the manual (they are described in some of the Haizea scientific publications, but these might be hard to swallow). Future versions of the Haizea manual will include a description of the main scheduling algorithms used, to better orient your choice of scheduling options. Finally, this chapter also covers how to run multiple unattended simulations.

5.1 Unattended simulations

To run Haizea as an unattended simulation requires setting the following options in the configuration file:

```
[general]
...
mode: simulated
...

[simulation]
...
clock: simulated
...
```

Additionally, the starting time of the simulation must be specified, along with a stopping condition:

```
[simulation]
...
starttime: 2006-11-25 13:00:00
stop-when: all-leases-done |
           besteffort-submitted |
           besteffort-done
...
```

5.2 Interactive simulations

To run Haizea as an interactive simulation, the following options must be set in the configuration file:

```
[general]
...
mode: simulated
...

[simulation]
...
clock: real
...
```

5.3 Specifying the simulated physical resources

The simulated physical resources are specified using the `resources` option in the `[simulation]` section. This option can take two values, "in-tracefile", which means that the description of the simulated site is in the tracefile, or a string specifying the site's resources. For the former, see Appendix C for details on how the simulated site is specified in the tracefile. When using the latter, the format of the string is:

```
<numnodes> <resource_type>:<resource_quantity>[,<resource_type>:<resource_quantity>]*
```

For example:

```
[simulation]
...
resources: 4  CPU:100 Memory:1024
...
```

The above describes a site with four nodes, each with one CPU and 1024 MB of memory. Note that you must always specify at least the "CPU" and "Memory" resource types.

5.4 Scheduling options

The scheduling options control how leases are assigned to resources.

5.4.1 Scheduling policies

Haizea includes a policy decision module that supports "pluggable policies", allowing developers to write their own scheduling policies. This is described in more detail in Chapter 8, and we describe here only the built-in policies that are included with Haizea.

The first policy is lease admission, which controls what leases are accepted by Haizea. Take into account that this decision takes place before Haizea even attempts to schedule the lease (so, you can think of lease admission as "eligibility to be scheduled"). The two built-in policies are to accept all leases, and to accept all leases *except* advance reservations.

```
[scheduling]
...
policy-admission: accept-all | no-ARs | <custom policy>
...
```

The next policy is lease preemptability, or what leases can be preempted. The two built-in policies are to not allow any preemptions, and to allow all ARs to preempt other leases.

```
[scheduling]
...
policy-preemption: no-preemption | ar-preempts-everything | <custom policy>
...
```

Finally, the host selection policy controls how Haizea chooses what physical hosts to map VMs to. The two built-in policies are to choose nodes arbitrarily (i.e., "no policy"), or to apply a greedy policy that tries to minimize the number of preemptions. Currently, you should choose the greedy policy unless you really know what you're doing.

```
[scheduling]
...
policy-host-selection: no-policy | greedy | <custom policy>
...
```

5.4.2 Backfilling algorithms

NOTE: This section assumes that you are familiar with backfilling algorithms. We will try to include a brief, didactic, explanation of backfilling algorithms in future versions of the manual.

Haizea supports both aggressive and conservative backfilling:

```
[scheduling]
...
backfilling: off | aggressive | conservative
...
```

An exact number of allowed future reservations can also be specified:

```
[scheduling]
...
backfilling: intermediate
backfilling-reservations: 4
...
```

5.4.3 Lease suspension and migration

Lease suspension can be allowed for all leases, only for 1-node leases ("serial" leases), or not allowed at all. Additionally, Haizea can schedule suspensions and resumptions to be locally or globally exclusive:

```
[scheduling]
...
suspension: none | serial-only | all
...
```

When suspending or resuming a VM, the VM's memory is dumped to a file on disk. To correctly estimate the time required to suspend a lease with multiple VMs, Haizea makes sure that no two suspensions/resumptions happen at the same time (e.g., if eight memory files were being saved at the same time to disk, the disk's performance would be reduced in a way that is not as easy to estimate as if only one file were being saved at a time).

Depending on whether the files are being saved to/read from a global or local filesystem, this exclusion can be either global or local:

```
[scheduling]
...
suspendresume-exclusion: local | global
...
```

When allocating time for suspending or resuming a single virtual machine with M MB of memory, and given a rate R MB/s of read/write disk throughput, Haizea will estimate the suspension/resumption time to be $\frac{M}{R}$. The `suspendresume-rate` option is used to specify R:

```
[simulation]
...
suspendresume-rate: 32
...
```

Lease migration can be disallowed, allowed, or allowed but without having to transfer any files from one to another:

```
[scheduling]
...
migration: no | yes | yes-notransfer
...
```

5.4.4 Lease preparation scheduling

Before a lease can start, it may require some preparation, such as transferring a disk image from a repository to the physical node where a VM will be running. When no preparation is necessary (e.g., assuming that all required disk images are predeployed on the physical nodes), the `lease-preparation` option must be set to `unmanaged`:

```
[general]
...
lease-preparation: unmanaged
...
```

When disk images are located in a disk image repository, Haizea can schedule the file transfers from the repository to the physical nodes to make sure that images arrive on time (when a lease has to start at a specific time) and to minimise the number of transfers (by reusing images on the physical nodes). To do this, `lease-preparation` option must be set to `imagetransfer`, we need to specify the network bandwidth of the image repository (in Mbits per second), and specify several options in the `[deploy-imagetransfer]` section:

```
[general]
...
lease-preparation: imagetransfer
...

[simulation]
...
imagetransfer-bandwidth: 100
...

[deploy-imagetransfer]
...
\# Image transfer scheduling options
...
```

Transfer mechanisms

The transfer mechanism specifies how the images will be transferred from the repository to the physical nodes. Haizea supports a unicast or a multicast transfer mechanism:

```
[deploy-imagetransfer]
...
transfer-mechanism: unicast | multicast
...
```

Whe using a unicast transfer mechanism, one image can only be transferred to one node at a time. When using multicast, it is possible to transfer the same image from the repository node to more than one physical node at the same time.

Avoiding redundant transfers

Haizea can take steps to detect and avoid redundant transfers (e.g., if two leases are scheduled on the same node, and they both require the same disk image, don't transfer the image twice; allow one to "piggyback" on the other). There is generally no reason to avoid redundant transfers.

```
[deploy-imagetransfer]
...
avoid-redundant-transfers: True | False
...
```

Disk image reuse

Haizea can create disk image caches on the physical nodes with the goal of reusing frequent disk images and reducing the number of transfers:

```
[deploy-imagetransfer]
...
diskimage-reuse: image-caches
diskimage-cache-size: 20000
...
```

5.4.5 The scheduling threshold

To avoid thrashing, Haizea will not schedule a lease unless all overheads can be correctly scheduled (which includes image transfers, suspensions, etc.). However, this can still result in situations where a lease is prepared, and then immediately suspended because of a blocking lease in the future. The scheduling threshold factor can be used to specify that a lease must not be scheduled unless it is guaranteed to run for a minimum amount of time (the rationale behind this is that you ideally don't want leases to be scheduled if they're not going to be active for at least as much time as was spent in overheads).

The default value is 1, meaning that the lease will be active for at least as much time t as was spent on overheads (e.g., if preparing the lease requires 60 seconds, and we know that it will have to be suspended, requiring 30 seconds, Haizea won't schedule the lease unless it can run for at least 90 minutes). In other words, a scheduling factor of F required a minimum duration of $F \cdot t$. A value of 0 could lead to thrashing, since Haizea could end up with situations where a lease starts and immediately gets suspended.

```
[scheduling]
...
scheduling-threshold-factor: 1
...
```

5.5 Running multiple unattended simulations

Haizea's configuration file allows for, at most, one tracefile to be used. However, when running simulations, it is often necessary to run through multiple tracefiles in a variety of configurations to compare the results of each tracefile/configuration combination. The "multi-configuration "file allows you to easily do just this. It is similar to the regular configuration file (all the options are the same), but it allows you to specify multiple tracefiles and multiple configuration profiles.

The multi-configuration file must contain a section called "`multi`" where you must specify the following:

- The tracefiles you want to use
- The "injected tracefiles" you want to use. In our own experiments, we found that it was easier to create workloads starting from a base workload and then "injecting" different types of workloads (e.g., a base workload of best-effort leases where AR leases of varying characteristics are "injected"). You can, of course, not specify any injected tracefiles.
- The directory where Haizea should store all the information it collects during the simulation (scheduling metrics, utilization information, etc.)

The `[multi]` section should look like this:

```
[multi]
tracedir: Directory with tracefiles
tracefiles: Tracefiles to use in experiments
injectiondir: Directory with injectable tracefiles
injectionfiles: Injectable tracefiles
basedatadir: Directory where raw data will be saved
```

Next, for each section you would ordinarily include in a regular configuration file, you can include common options (shared by all profiles) and profile-specific options. For example, assuming you want to specify options in the `general` and `simulation` sections, and you want to create two profiles called `nobackfilling` and `withbackfilling`, you would have to create the following sections:

```
[common:general]
...

[common:simulation]
...

[nobackfilling:general]
...

[nobackfilling:simulation]
...

[withbackfilling:general]
...

[withbackfilling:simulation]
...
```

An example multi-configuration file is provided in `/usr/share/haizea/etc/sample-multi.conf`. Using this file, or once you've created your own, you can use the `haizea-generate-configs` to create the individual configuration files (one for every combination of tracefile, injected tracefile, and profile):

```
haizea-generate-configs -c config -d dir
```

The `-c` parameter is used to specify the multi-config file, and the `-d` parameter is used to specify where the configuration files should be created. Since running each configuration individually would be cumbersome, you can also use the `haizea-generate-script` command to generate a script that will run through all the generated configuration files. This command requires Mako Templates for Python, so make sure you install Mako before using `haizea-generate-scripts`. Haizea currently includes two script templates: one to generate a BASH script that will call haizea with each individual configuration file, and one to generate a basic Condor submission script. For example, to generate the BASH script, you would run the command like this:

```
haizea-generate-scripts -c config -d dir -t /usr/share/haizea/etc/run.sh.template
```

6 Haizea and OpenNebula

OpenNebula (`http://www.opennebula.org/`) is a virtual infrastructure manager that enables the dynamic deployment and re-allocation of virtual machines on a pool of physical resources. Haizea can be used to extend OpenNebula's scheduling capabilities, allowing it to support advance reservation of resources and queueing of best effort requests. OpenNebula and Haizea complement each other, since OpenNebula provides all the enactment muscle (OpenNebula can manage Xen, KVM, and VMWare VMs on a cluster) and Haizea provides the scheduling brains. Using both of them together is simple, since Haizea acts as a drop-in replacement for OpenNebula's scheduling daemon.

This chapter explains how to use OpenNebula and Haizea together, and explains how to submit requests to OpenNebula to use Haizea's scheduling capabilities.

6.1 Installing OpenNebula and Haizea

If you have not already done so, you will need to install OpenNebula 1.4 and the latest version of Haizea. Start by installing OpenNebula, and then installing Haizea.

Before proceeding, you may want to follow the OpenNebula quickstart guide (`http://www.opennebula.org/doku.php?id=documentation:rel1.4:qg`) to verify that your OpenNebula installation is working fine. The rest of this document assumes that OpenNebula is correctly installed, and that you know what a *virtual machine template* is ("VM templates" is how VMs are requested to OpenNebula, so we'll be working with them quite a bit). You may also want to follow the Haizea Quickstart Guide (see Chapter 4, to verify that Haizea is correctly installed.

6.2 Configuring Haizea

Haizea must be configured to run in OpenNebula mode. Haizea includes a sample OpenNebula configuration file that you can use as a starting point. This file is installed, by default, in `/usr/share/haizea/etc/sample_opennebula.conf` (there is also a `sample_opennebula_barebones.conf` file that has the same options, but without any documentation). In OpenNebula mode, Haizea will process requests coming from OpenNebula, and will send all enactment commands to OpenNebula. To activate this mode, the `mode` option of the `general` section in the Haizea configuration file must be set to `opennebula`:

```
[general]
...
mode: opennebula
...
```

Haizea interacts with OpenNebula through it's XML-RPC API, so you need to tell Haizea what host OpenNebula is on. This is done in the `opennebula` section:

```
[opennebula]
# Typically, OpenNebula and Haizea will be installed
# on the same host, so the following option should be
# set to 'localhost'. If they're on different hosts,
# make sure you modify this option accordingly.
host: localhost
```

Additionally, if OpenNebula is not listening on its default port (2633), you can use the `port` option in the `opennebula` section to specify a different port.

There are also a couple options in the `scheduling` section that are relevant to OpenNebula mode, but which you do not need to concern yourself with yet (they are described at the end of this chapter).

6.3 Running OpenNebula and Haizea together

Now that Haizea is configured to run alongside OpenNebula, running them is as simple as starting the OpenNebula daemon:

```
oned
```

Followed by Haizea:

```
haizea -c /usr/share/haizea/etc/sample_opennebula.conf
```

The above assumes that you are running OpenNebula and Haizea in the same machine and with the same user. If this is not the case, you have to set the `ONE_AUTH` environment variable (as described in the OpenNebula documentation) for the user and machine running Haizea. The variable must contain the username and password of the OpenNebula administrator user (typically called `oneadmin`) with the format `username:password`.

By default, Haizea runs as a daemon when running in OpenNebula mode. For this chapter, you may want to run it in the foreground so you can see the Haizea log messages in your console:

```
haizea --fg -c /usr/share/haizea/etc/sample_opennebula.conf
```

When Haizea starts up, it will print out something like this:

```
[                      ] ENACT.ONE.INFO Fetched N nodes from OpenNebula
[2009-07-30 18:36:54.07] RM      Starting resource manager
[2009-07-30 18:36:54.07] RPCSERVER RPC server started on port 42493
[2009-07-30 18:36:54.07] CLOCK   Starting clock
```

This means that Haizea has correctly started up, contacted OpenNebula and detected that there are N physical nodes (the value of N will depend, of course, on how many nodes you have in your system).

Haizea is a drop-in replacement for OpenNebula's default scheduler (`mm_sched`). Do not run Haizea and `mm_sched` at the same time, or funny things will happen.

6.4 A quick test

At this point, OpenNebula and Haizea are both running together, and waiting for you to submit a VM request. From the user's perspective, you will still be submitting your requests to OpenNebula, and Haizea will do all the scheduling work backstage. However, you will be able to add an `HAIZEA` parameter to your OpenNebula request to access Haizea's features.

So, to test that OpenNebula and Haizea are working correctly, start by taking a known-good OpenNebula template. Just to be on the safe side, you may want to try it with the default scheduler first, to make sure that the VM itself works correctly, etc. Then, just add the following parameter to the template:

```
HAIZEA = [
  start        = "+00:00:30",
  duration     = "00:01:00",
  preemptible  = "no"
]
```

The exact meaning of these parameters is explained later on in this document. In a nutshell, the values specified above tell Haizea to schedule the VM to start exactly 30 seconds in the future, to run for one minute, and to not allow the allocated resources to be preempted by other requests. This corresponds to an Haizea *advance reservation lease* (see Chapter 2).

Before you submit your request to OpenNebula, take a look at the Haizea log. You should see something like this repeating every minute:

```
[2009-07-30 18:38:44.00] CLOCK   Waking up to manage resources
[2009-07-30 18:38:44.00] CLOCK   Wake-up time recorded as 2009-07-30 18:38:44.00
[2009-07-30 18:38:44.01] CLOCK   Going back to sleep.
                                 Waking up at 2009-07-30 18:38:54.00
                                 to see if something interesting has happened by then.
```

Haizea is configured, by default, to ask OpenNebula if there are any pending requests every minute. Since you haven't submitted anything, Haizea just wakes up every minute and goes right back to sleep. So, go ahead and submit your request (the one where you added the HAIZEA parameter). Assuming you named the template ar.one, run the following:

```
onevm submit ar.one
```

If you run `onevm list` to see the VMs managed by OpenNebula, you'll see that the request is in a `pending` state:

```
  ID     USER     NAME STAT CPU     MEM        HOSTNAME        TIME
-------------------------------------------------------------------
  42    borja     test pend   0       0                00 00:00:02
```

Next time Haizea wakes up, you should see something like this:

```
[2009-07-30 18:41:49.16] CLOCK   Waking up to manage resources
[2009-07-30 18:41:49.16] CLOCK   Wake-up time recorded as 2009-07-30 18:41:49.00
[2009-07-30 18:41:49.19] LSCHED  Lease #1 has been requested.
[2009-07-30 18:41:49.19] LSCHED  Lease #1 has been marked as pending.
[2009-07-30 18:41:49.19] LSCHED  Scheduling AR lease #1, 1 nodes
                                     from 2009-07-30 18:42:15.00
                                       to 2009-07-30 18:43:15.00.
[2009-07-30 18:41:49.19] LSCHED  AR lease #1 has been scheduled.

[2009-07-30 18:41:49.19] CLOCK   Going back to sleep.
                                 Waking up at 2009-07-30 18:42:15.00
                                 to handle slot table event.
```

Notice how Haizea detected that OpenNebula had an AR request, and then scheduled it to start 30 seconds in the future. In fact, Haizea takes care to wake up at that time so the VM can start at exactly that time.

If you run ***onevm list****, the request will still be shown as* ***pending****. OpenNebula doesn't track Haizea's internal states, so it will consider the request "pending" until Haizea starts up the VM. You can check the state of Haizea leases using the* ***haizea-list-leases*** *command.*

Currently, Haizea has to poll OpenNebula every minute to ask if there are any new requests. An upcoming version of Haizea will support an event-based model where OpenNebula can send Haizea a notification as soon as a new request is received (so the user doesn't have to wait until the next time Haizea wakes up to process the request).

When the VM is scheduled to start, you will see the following in the Haizea logs:

```
[2009-07-30 18:42:15.02] CLOCK   Waking up to manage resources
[2009-07-30 18:42:15.02] CLOCK   Wake-up time recorded as 2009-07-30 18:42:15.00
[2009-07-30 18:42:15.04] VMSCHED Started VMs for lease 1 on nodes [2]
[2009-07-30 18:42:15.09] CLOCK   Going back to sleep.
                                 Waking up at 2009-07-30 18:43:00.00
                                 to handle slot table event.
```

Haizea has instructed OpenNebula to start the VM for the advance reservation. If you run `onevm list`, the VM will now show up as running:

```
  ID     USER     NAME STAT CPU     MEM        HOSTNAME        TIME
-------------------------------------------------------------------
  42    borja     test runn  10   65536       cluster05 00 00:00:52
```

You should be able to access the VM (if you configured it with networking and SSH). However, since we requested the VM to run for just a minute, you will soon see the following in the Haizea logs:

```
[2009-07-30 18:43:00.04] CLOCK   Waking up to manage resources
[2009-07-30 18:43:00.04] CLOCK   Wake-up time recorded as 2009-07-30 18:43:00.00
[2009-07-30 18:43:00.05] VMSCHED Stopped VMs for lease 1 on nodes [2]
[2009-07-30 18:43:05.07] CLOCK   Going back to sleep.
                                 Waking up at 2009-07-30 18:43:15.00
                                 to handle slot table event.

[2009-07-30 18:43:15.00] CLOCK   Waking up to manage resources
[2009-07-30 18:43:15.00] CLOCK   Wake-up time recorded as 2009-07-30 18:43:15.00
[2009-07-30 18:43:15.00] VMSCHED Lease 1's VMs have shutdown.
[2009-07-30 18:43:15.01] CLOCK   Going back to sleep.
                                 Waking up at 2009-07-30 18:44:15.00
                                 to see if something interesting has happened by then.
```

6.5 The `HAIZEA` parameter in OpenNebula

The previous section showed how you can add an `HAIZEA` parameter to your OpenNebula VM template to request a simple advance reservation. The three Haizea options (`start`, `duration`, and `preemptible`) can take other values:

- `start`: This option specifies when the VM will start. Valid values are:
  - `best_effort`: The VM will be scheduled as soon as resources are available. If resources are not available right now, the request is put on a queue and it remains there until there are sufficient resources (requests are scheduled on a first-come-first-serve basis).
  - `now`: The VM will be scheduled right now. If resources are not available right now, the request is rejected.
  - Exact ISO timestamp: i.e., `YYYY-MM-DD HH:MM:SS`. The VM must start at exactly that time. If enough resources are not available at that time, the resource is not requested.
  - Relative ISO timestamp: For convenience's sake (and also for testing) this provides an easy way of specifying that a VM must start "at T time after the VM is submitted". The format would be `+HH:MM:SS` (the "`+`" is not ISO, but is used by Haizea to determine that it is a relative timestamp)
- `duration`: The duration of the lease. Possible values are:
  - `unlimited`: The lease will run forever, until explicitly stopped
  - ISO-formatted time: i.e., `HH:MM:SS`
- `preemptible`: This option can be either yes or no.
- `group`: This option can take on any string value, and allows you to schedule several VMs as a group (or, in Haizea terminology, as a single lease with multiple nodes). All OpenNebula VM templates with the same group name will be considered part of the same lease (i.e., all the VMs will be scheduled in a all-or-nothing fashion: all VMs must be able to start/stop at the same time). Future versions of OpenNebula will automatically manage this option, so users don't have to worry about manually setting this option in multiple VM templates (which can be error-prone).

Usually, you will want to use these options to create one of Haizea's supported lease types:

6.5.1 Advance reservations

When you need your VM available at a specific time, this is called an advance reservation, or AR. The VM we used above is an example of an AR:

```
HAIZEA = [
  start        = "+00:00:30",
  duration     = "00:01:00",
  preemptible  = "no"
]
```

Of course, instead of specifying that you want your VM to start after a certain amount of time has passed (30 seconds, in this case), you can also specify an exact start time:

```
HAIZEA = [
  start        = "2008-11-04 11:00:00",
  duration     = "03:00:00",
  preemptible  = "no"
]
```

NOTE: Haizea currently only supports non-preemptible ARs.

6.5.2 Best-effort provisioning

When you instruct Haizea to determine the start time on a best-effort basis, your request will be allocated resources as soon as they become available. Take into account that your request may be placed on a queue, and you'll have to wait until your turn is up. You can use the `haizea-list-leases` and `haizea-show-queue` to check on the state of your lease.

```
HAIZEA = [
  start        = "best_effort",
  duration     = "01:00:00",
  preemptible  = "yes"
]
```

A best-effort VM can be preemptible or non-preemptible. If you request a non-preemptible VM, you may still have to wait in the queue until you get your resources but, once you do, no one can take them from you.

6.5.3 Immediate provisioning

Sometimes, you need a VM right now or not at all. In that case, you can set the starting time to `now`.

```
HAIZEA = [
  start        = "now",
  duration     = "unlimited",
  preemptible  = "no"
]
```

6.6 Additional OpenNebula configuration options

When running Haizea with OpenNebula, you must specify at least the `host` option in the `[opennebula]` section of the configuration file. However, there are additional options in other sections that you can tweak:

6.6.1 Wakeup interval

This is the interval, in seconds, at which Haizea will wake up to process pending requests in OpenNebula. The default is 60 seconds.

```
[scheduling]
...
wakeup-interval: 60
...
```

6.6.2 Suspend/resume rate interval

This option provides Haizea with an estimate of how long it takes for OpenNebula to suspend or resume a virtual machine. This is estimated in MB per second, and is largely dependent on the disk read/write transfer speeds on your system (so, if a VM has 1024 MB of memory, and the suspend rate is estimated to be 64MB/s, Haizea will estimate that suspension will take 16 seconds). If you do not specify a value, Haizea will conservatively assume a rate of 32MB/s. A good estimate will allow Haizea to more correctly schedule resources, but an incorrect estimate will not result in an error (although a warning will be noted in the logs).

```
[scheduling]
...
suspend-rate: 32
resume-rate: 32
...
```

Additionally, since OpenNebula currently only supports suspending to a global filesystem (i.e., the RAM file created when suspending a VM is saved to a global filesystem, such as an NFS drive), you will need to specify that suspensions and resumptions must be globally exclusive (to make sure that no more than one RAM file is being saved to the global filesystem at any one time). You can control this using the `suspendresume-exclusion` option in the `[scheduling]` section:

```
[scheduling]
...
suspendresume-exclusion: global
...
```

This option is set to `global` in the sample OpenNebula configuration file, but defaults to `local` when not specified.

6.6.3 Non-schedulable interval

The minimum amount of time that must pass between when a request is scheduled to when it can actually start (i.e., this makes sure that the scheduling function doesn't make reservations with starting times that will be in the past by the time the scheduling function ends). The default (10 seconds) should be good for most configurations, but may need to be increased if you're dealing with exceptionally high loads.

```
[scheduling]
...
non-schedulable-interval: 10
...
```

6.7 Known issues and limitations

The following are known issues and limitations when using Haizea with OpenNebula:

- As pointed out in this guide, Haizea has to poll OpenNebula every minute to ask if there are any new requests. Although OpenNebula 1.4 added a "hook mechanism" that allows actions to be carried out when certain events happen (such as sending Haizea notifications of a VM that has died, a suspend operation that finished before expected, etc.), Haizea currently does not use this hook mechanism.
- Haizea currently cannot do any image deployment with OpenNebula, and VM images are assumed to be predeployed on the physical nodes, or available on a shared NFS filesystem. Although OpenNebula includes support for interfacing with a *transfer manager* to handle various VM deployment scenarios, Haizea currently does not access this functionality.
- Haizea cannot enact cold migrations in OpenNebula (i.e., migrating a suspended VM to a different node if resources become available earlier on a different node than the one where the VM was suspended on). Haizea actually has all the scheduling code for this, and only the enactment "glue" is missing.

7 Analysing scheduling data

While Haizea is running, it collects data that can be analysed offline (accepted/rejected leases, waiting times, etc.). This data is saved to disk when Haizea stops running so, for now, this information is (in practice) only useful for simulation experiments. In the future, Haizea will save data periodically to disk so it can also be analysed online.

The information that is collected can be specified through a series of *probes*. For example, there is a `best-effort` probe that collects information relevant to best-effort leases, such as the time the lease had to wait in the queue until it was allocated resources. Haizea includes several probes (see Appendix D) and also allows you to write your own probes (see Chapter 9)

The file where the collected data will be saved and the probes to use are specified in the `[accounting]` section of the configuration file:

```
[accounting]
datafile: /var/tmp/haizea.dat
probes: ar best-effort immediate utilization
```

This file is not human-readable, and there are two ways of accessing its contents: using the `haizea-convert-data` command or programmatically through Python. Both are described in this chapter.

7.1 Type of data collected

Accounting probes can collect three types of data:

Per-lease data : Data attributable to individual leases or derived from how each lease was scheduled. For example, as mentioned earlier, the `best-effort` probe collects the waiting time of each best-effort lease.

Per-run data : Data from a single run of Haizea. For example, the `ar` probe collects the total number of AR leases that were accepted and rejected during the entire run.

Counters : A counter is a time-ordered list showing how some metric varied throughout a single run of Haizea. For example, the `best-effort` probe keeps track of the length of the queue.

See Appendix D) for a list of probes included with Haizea, and a description of the specific data they collect.

7.2 The `haizea-convert-data` command

The `haizea-convert-data` command will convert the contents of the data file into a CSV file.

To print out all the per-lease data, simply run the following:

```
haizea-convert-data -t per-lease /var/tmp/haizea.dat
```

This will print out one line per lease, showing its lease ID and all data collected for that lease. Take into account that some fields will be empty, as a probe might collect data just for one specific type of lease (e.g., AR leases will have empty values for the 'Waiting time' information collected by the `best-effort`).

To print out all the per-run data, run the following:

```
haizea-convert-data -t per-run /var/tmp/haizea.dat
```

To print out a counter, run the following:

```
haizea-convert-data -t counter -c countername /var/tmp/haizea.dat
```

Where `countername` should be substituted for the counter you want to access. If you do not know what counters are included in the file, the following will print out a list of counters:

```
haizea-convert-data -l /var/tmp/haizea.dat
```

When running multiple simulations (as described in 5.5), Haizea will generate one data file for each simulation profile, which are all stored in the same directory. `haizea-convert-data` can also be used to produce aggregate statistics from all these data files. For example:

```
haizea-convert-data -t per-run /var/tmp/results/*.dat
```

This will print out one line per simulation run, each with the per-run data for the run along with the simulation profile, tracefile, and injection file used in that run. Similarly, you can run `haizea-convert-data` with `-t per-lease` or `-t counter` to print the per-lease data or a counter from multiple simulation runs, using the simulation profile, tracefile, and injection file columns to disambiguate the run the data originated from.

7.3 Analysing data programmatically

The data file generated by Haizea is a Python-pickled `AccountingData` object. This object contains all the per-lease and per-run data, along with all the counters. You can analyse the data programmatically by unpickling the file from your own Python code and accessing the data contained in the `AccountingData` object (see the generated pydoc documentation linked from the Haizea website for details on the object's attributes). An example of how this file is unpickled, and some of its information accessed, can be found in function `haizea_convert_data` in module `haizea.cli.commands`.

Part III

Customizing Haizea

8 Writing your own policies

Haizea uses several scheduling algorithms internally to determine what resources to allocate to a lease. For the most part, modifying these algorithms requires digging deep into the Haizea code. However, several scheduling decisions that depend on an organizations own resource allocation policies are factored out of the main scheduling code into pluggable *policy decision module*. In particular, the following decisions are factored out:

Lease admission : Should a lease request be accepted or rejected? Take into account that this decision takes place before Haizea determines if the request is even feasible. For example, an organization may require that all AR leases must be requested at least one hour in advance, regardless of whether there would be enough resources to satisfy the request before that time. However, being accepted doesn't guarantee the lease will get resources (although this could factor into the decision too); an AR lease could meet the "one hour advance warning" requirement, but still end up being rejected because there are no resources available at the requested time.

Lease preemptability : How preemptable is a lease? Not all leases are created equal and, if the scheduler determines that a lease request can only be satisfied by preempting other leases, it may have to determine what leases are better candidates for preemption. For example, given a choice of preempting a lease that's been running for a week and another that's been running for five minutes, an organization might prefer to not interrupt the long-running lease.

Host selection : What hosts should a lease be scheduled in? When the scheduler has a choice of several physical hosts on which to deploy VMs, some might be preferable than others. For example, an organization might want to pack as many VMs into the same hosts, to shut down those that are not running VMs, while another might want to spread those VMs across several hosts, leaving some free resources available in each host in case the VMs need extra capacity further down the road.

As you can see, these are all policy decisions that are driven by an organization's own goals for its resources. Thus, Haizea makes it simple to write your own policy decision code *without* having to modify Haizea's code. All you have to do is write a simple Python module, and then "plug it" into Haizea by instructing it (through the configuration file) to use that module. This chapter describes how this is done.

This documentation refers to Haizea objects, such as `Lease` *and* `SlotTable` *that are not yet documented in this manual. For now, you will need to read the Haizea Pydoc documentation (linked from the Documentation section of the Haizea website) to see what attributes and methods these classes have. A more complete documentation will be included in the final 1.0 release.*

8.1 Lease admission

A lease admission policy module looks like this:

```
from haizea.core.scheduler.policy import LeaseAdmissionPolicy

class MyPolicy(LeaseAdmissionPolicy):
    def __init__(self, slottable):
        LeaseAdmissionPolicy.__init__(self, slottable)

    def accept_lease(self, lease):
        # Your code goes here
```

The `accept_lease` method receives a `Lease` object, and must return `True` if the lease can be accepted, and `False` if it should be rejected. You can also add code to the constructor, but cannot alter its parameter list. Haizea includes some built-in admission policies that you can see in `src/haizea/policies/admission.py`

The lease admission policy that Haizea must use is specified using the `policy-admission` option of the `[scheduling]` section in the configuration file. So, assuming you save your module as `policies.py`, you would specify the following in the configuration file:

```
[scheduling]
...
policy-admission: policies.MyPolicy
...
```

For this to work, you have to make sure that the `policies.py` module you created is in your `PYTHONPATH` when you start Haizea.

For example, let's suppose we want to write an admission policy that, as described earlier, will reject AR leases that are not requested at least one hour in advance. This policy module would look like this:

```
from haizea.core.scheduler.policy import LeaseAdmissionPolicy
from haizea.core.leases import Lease
from haizea.common.utils import get_clock
from mx.DateTime import TimeDelta

class MyPolicy(LeaseAdmissionPolicy):
    def __init__(self, slottable):
        LeaseAdmissionPolicy.__init__(self, slottable)

    def accept_lease(self, lease):
        allowed = TimeDelta(hours=1)
        now = get_clock().get_time()

        if lease.get_type() == Lease.ADVANCE_RESERVATION:
            if lease.start.requested - now <= allowed:
                return False
        return True
```

Save this file as `policies.py`, make sure the directory it's in is in your `PYTHONPATH`, and set `[scheduling].policy-admission` to `policies.MyPolicy` in the configuration file. If you rerun the example from the quickstart guide, instead of seeing this:

```
[2006-11-25 13:15:00.00] LSCHED  Lease #2 has been requested.
[2006-11-25 13:15:00.00] LSCHED  Lease #2 has been marked as pending.
```

You will see that the AR lease, which is requested 15 minutes before it starts, is rejected:

```
[2006-11-25 13:15:00.00] LSCHED  Lease #2 has been requested.
[2006-11-25 13:15:00.00] LSCHED  Lease #2 has not been accepted
```

In fact, if you modify the starting time to be the following:

```
<start>
  <exact time="02:00:00"/>
</start>
```

The lease will be accepted again, although it will start later than before:

```
[2006-11-25 15:00:00.00] VMSCHED Started VMs for lease 2 on nodes [1, 2, 3, 4]
[2006-11-25 15:30:00.00] VMSCHED Stopped VMs for lease 2 on nodes [1, 2, 3, 4]
```

8.2 Lease preemptability

A lease preemptability policy module looks like this:

```
from haizea.core.leases import Lease
from haizea.core.scheduler.policy import PreemptabilityPolicy

class MyPolicy(PreemptabilityPolicy):
    def __init__(self, slottable):
        PreemptabilityPolicy.__init__(self, slottable)

    def get_lease_preemptability_score(self, preemptor, preemptee, time):
        # Your code goes here
```

The `get_lease_preemptability_score` receives two `Lease` objects, the lease that wants to preempt resources (the `preemptor`) and the lease that is being considered for preemption (the `preemptee`), and the time at which the preemption would take place. The method should return the *preemptability score* of the preemptee, indicating how preemptable the lease is. This score can take on the following values:

- -1: Cannot be preempted under any circumstances
- $0.0 \leq \text{score} \leq 1.0$: The lease can be preempted. The higher the score, the "more preemptable" it is. Take into account that this is a relative measure: the score will be used by the scheduler to determine which of several leases is a better candidate for preemption.

The lease preemptability policy to use is specified using the `policy-preemption` option of the `[scheduling]` section in the configuration file. So, assuming you save your module as `policies.py`, you would specify the following in the configuration file:

```
[scheduling]
...
policy-preemption: policies.MyPolicy
...
```

8.3 Host selection

A host selection policy module looks like this:

```
from haizea.core.scheduler.policy import HostSelectionPolicy

class NoPolicy(HostSelectionPolicy):
    def __init__(self, slottable):
        HostSelectionPolicy.__init__(self, slottable)


    def get_host_score(self, node, time, lease):
        # Your code goes here
```

The `get_host_score` method receives a physical host (the integer node identifier used in the slot table, which all policy modules have access to), a time, and a `Lease` object we would like to schedule at that time. This method returns a score indicating how desirable that host is for that lease at that time. The score can be between 0.0 and 1.0, and the higher the score, the "more desirable" the physical host is. Like the lease preemptability score, this is a relative measure; the score will be used to determine which of several physical hosts is more desirable for this lease.

The host selection policy to use is specified using the `policy-host-selection` option of the `[scheduling]` section in the configuration file. So, assuming you save your module as `policies.py`, you would specify the following in the configuration file:

```
[scheduling]
...
policy-host-selection: policies.MyPolicy
...
```

9 Writing accounting probes

In Chapter 7 we saw that Haizea collects data while running through the use of *probes*. While Haizea includes several probes, it is also possible for you to write your own probes by implementing a class that extends from the `AccountingProbe` class. A barebones probe would look like this:

```
from haizea.core.accounting import AccountingProbe

class MyProbe(AccountingProbe):

    def __init__(self, accounting):
        AccountingProbe.__init__(self, accounting)
        # Create counters, per-lease stats, per-run stats

    def finalize_accounting(self):
  # Collect information

    def at_timestep(self, lease_scheduler):
  # Collect information

    def at_lease_request(self, lease):
  # Collect information

    def at_lease_done(self, lease):
  # Collect information
```

All the methods shown above are also present in `AccountingProbe`, but don't do anything. You have to override some or all of the methods to make sure that data gets collected. More specifically:

`at_timestep` Override this method to perform any actions every time the Haizea scheduler wakes up. The `lease_scheduler` parameter contains Haizea's lease scheduler (an instance of the `LeaseScheduler` class), which you can use to gather scheduling data.

`at_lease_request` Override this method to collect data after a lease has been requested.

`at_lease_done` Override this method to collect data after a lease is done (this includes successful completion and rejected/cancelled/failed leases).

`finalize_accounting` Override this method to perform any actions when data collection stops. This is usually where per-run data is computed.

Probes can collect three types of data:

Per-lease data: Data attributable to individual leases or derived from how each lease was scheduled.

Per-run data: Data from an entire run of Haizea

Counters: A counter is a time-ordered list showing how some metric varied throughout a single run of Haizea.

The probe's constructor should create the counters and specify what per-lease data and per-run data will be collected by your probe (the methods to do this are described next). Notice how a probe's constructor receives a `accounting` parameter. When creating your probe, Haizea will pass an `AccountingDataCollection` object that you will be able to use in your probe's other methods to store data.

Once a probe has been implemented, it can be used in Haizea by specifying its full name in the `probes` option of the `accounting` section of the configuration file. For example, suppose you created a class called `MyProbe` in the foobar.probes module. To use the probe, the `probes` option would look like this:

```
probes: foobar.probes.MyProbe
```

When running Haizea, you have to make sure that `foobar.probes.MyProbe` is in your `PYTHONPATH`. After running Haizea with your probe, you can access the data it collects using the `haizea-convert-data` command described in Section 7.2

9.1 Collecting per-lease data

To collect per-lease data, you first have to specify the new type of data (or "stat") you will be collecting in your probe's constructor. This is done using the `create_lease_stat` method in `AccountingDataCollection` (which is stored in an `accounting` attribute in all probes). For example, let's assume you want to keep track of an admittedly silly statistic: whether the lease's identifier is odd or even. You could create a stat called `Odd or even?`:

```
from haizea.core.accounting import AccountingProbe

class MyProbe(AccountingProbe):

    def __init__(self, accounting):
        AccountingProbe.__init__(self, accounting)
        self.accounting.create_lease_stat("Odd or even?")
```

To set the value of this stat, you must use the `set_lease_stat` method in `AccountingDataCollection`. For this stat, it would make sense to call this method from the `at_lease_request` method in the probe:

```
def at_lease_request(self, lease):
    if lease.id \% 2 == 1:
        value = "odd"
    else:
        value = "even"
    self.accounting.set_lease_stat("Odd or even?", lease.id, value)
```

If you run Haizea with this probe, and then use `haizea-convert-data` to print the per-lease data collected by the probes, there will be an additional column titled `Odd or even?` in the generated CSV file.

9.2 Collecting per-run data

Collecting per-run data is similar to collecting per-lease data, and relies on two methods in `AccountingDataCollection`: `create_stat` to create the stat in the probe constructor and `set_stat` to set the value of the stat. Given that per-run data summarizes information from the entire run, `set_stat` will usually be called from the probe's `finalize_accounting` method.

9.3 Creating and updating counters

To collect data using a counter you must first create the counter from the probe's constructor using the `create_counter` method in `AccountingDataCollection`:

```
create_counter(counter_id, avgtype)
```

The first parameter is the name of the counter. The second parameter specifies the type of average to compute for the counter; Counters can store not just the value of the counter throughout time, but also a running average. There are three types of averages that can be specified through the `avgtype`

`AccountingDataCollection.AVERAGE_NONE`: Don't compute an average

`AccountingDataCollection.AVERAGE_NORMAL`: For each entry, compute the average of all the values including and preceding that entry.

`AccountingDataCollection.AVERAGE_TIMEWEIGHTED`: For each entry, compute the average of all the values including and preceding that entry, weighing the average according to the time between each entry.

All counters are initialized to zero.
The counter can be updated using one of the following three methods:

```
incr_counter(counter_id, lease_id):
decr_counter(counter_id, lease_id):
append_to_counter(counter_id, value, lease_id):
```

All three methods receive the name of the counter (`counter_id`) and, optionally, the identifier of the lease that caused the update. `incr_counter` and `decr_counter` increment or decrement the counter, respectively, while `append_to_counter` changes the counter's value to the value specified by the `value` parameter.

9.4 Examples

See the `haizea.pluggable.accounting` module for the source code of the default probes included with Haizea.

Part IV

Appendices

A Command-line interface reference

A.1 haizea

This is the main Haizea command. By default, it will start Haizea as a daemon, which can receive requests via RPC or interact with other components such as OpenNebula. It can also start as a foreground process, and write all log messages to the console. All Haizea options are specified through the configuration file.

Option	Description
`-D, --debug`	Run command in debug mode.
`-c CONF, --conf=CONF`	The location of the Haizea configuration file. If not specified, Haizea will first look for it in /etc/haizea/haizea.conf and then in ~/.haizea/haizea.conf.
`-f, --fg`	Runs Haizea in the foreground.
`--stop`	Stops the Haizea daemon.

A.2 haizea-request-lease

Requests a new lease.

Option	Description
`-D, --debug`	Run command in debug mode.
`-s SERVER, --server=SERVER`	Haizea RPC server URI. If not specified, the default http://localhost:42493 is used
`-f FILE, --file=FILE`	File containing a lease description in XML.
`-t START, --start=START`	Starting time. Can be an ISO timestamp, "best_effort", or "now"
`-d DURATION, --duration=DURATION`	Duration. Can be an ISO timestamp or "unlimited"
`-n NUMNODES, --numnodes=NUMNODES`	Number of nodes.
`--preemptible`	Specifies a preemptible lease.
`--non-preemptible`	Specifies a non-preemptible lease.
`-c CPU, --cpu=CPU`	Percentage of CPU (must be 0 $<$ c $<=$ 100)
`-m MEM, --mem=MEM`	Memory per node (in MB)
`-i VMIMAGE, --vmimage=VMIMAGE`	Disk image identifier.
`-z VMIMAGESIZE, --vmimagesize=VMIMAGESIZE`	Disk image size.

A.3 `haizea-cancel-lease`

Cancel a lease.

Option	Description
`-D, --debug`	Run command in debug mode.
`-s SERVER, --server=SERVER`	Haizea RPC server URI. If not specified, the default http://localhost:42493 is used
`-l LEASE, --lease=LEASE`	ID of lease to cancel.

A.4 haizea-list-leases

List all active leases in the system (i.e., not including completed or cancelled leases)

Option	Description
`-D, --debug`	Run command in debug mode.
`-s SERVER, --server=SERVER`	Haizea RPC server URI. If not specified, the default http://localhost:42493 is used

A.5 haizea-show-queue

Show the contents of the Haizea queue

Option	Description
`-D, --debug`	Run command in debug mode.
`-s SERVER, --server=SERVER`	Haizea RPC server URI. If not specified, the default http://localhost:42493 is used

A.6 haizea-list-hosts

List hosts managed by Haizea

Option	Description
`-D, --debug`	Run command in debug mode.
`-s SERVER, --server=SERVER`	Haizea RPC server URI. If not specified, the default http://localhost:42493 is used

A.7 haizea-generate-configs

Takes an Haizea multiconfiguration file and generates the individual configuration files. See the Haizea manual for more details on multiconfiguration files.

Option	Description
`-D, --debug`	Run command in debug mode.
`-c CONF, --conf=CONF`	Multiconfiguration file.
`-d DIR, --dir=DIR`	Directory where the individual configuration files must be created.

A.8 haizea-generate-scripts

Generates a script, based on a script template, to run all the individual configuration files generated by haizea-generate-configs. This command requires Mako Templates for Python (http://www.makotemplates.org/).

Option	Description
`-D, --debug`	Run command in debug mode.
`-c CONF, --conf=CONF`	Multiconfiguration file used in haizea-generate-configs.
`-d CONFDIR, --confdir=CONFDIR`	Directory containing the individual configuration files.
`-t TEMPLATE, --template=TEMPLATE`	Script template (sample templates are included in /usr/share/haizea/etc)
`-m, --only-missing`	If specified, the generated script will only run the configurations that have not already produced a datafile. This is useful when some simulations fail, and you don't want to have to rerun them all.

A.9 haizea-convert-data

Converts Haizea datafiles into another (easier to process) format.

Option	Description
`-D, --debug`	Run command in debug mode.
`-t TYPE, --type=TYPE`	Type of data to produce.
`-f FORMAT, --format=FORMAT`	Output format. Currently supported: csv

B Configuration file reference

B.1 Section [general]

This section is used for general options affecting Haizea as a whole.
This section is required

B.1.1 Option loglevel

Valid values: STATUS, INFO, DEBUG, VDEBUG

Required: No (default is INFO)

Description: Controls the level (and amount) of log messages. Valid values are:

- STATUS: Only print status messages
- INFO: Slightly more verbose that STATUS
- DEBUG: Prints information useful for debugging the scheduler.
- VDEBUG: Prints very verbose information on the scheduler's internal data structures. Use only for short runs.

B.1.2 Option logfile

Valid values: Any string

Required: No (default is /var/tmp/haizea.log)

Description: When running Haizea as a daemon, this option specifies the file that log messages should be written to.

B.1.3 Option mode

Valid values: simulated, opennebula

Required: Yes

Description: Sets the mode the scheduler will run in. Currently the only valid values are "simulated" and "opennebula". The "simulated" mode expects lease requests to be provided through a trace file, and all enactment is simulated. The "opennebula" mode interacts with the OpenNebula virtual infrastructure manager (http://www.opennebula.org/) to obtain lease requests and to do enactment on physical resources.

B.1.4 Option lease-preparation

Valid values: unmanaged, imagetransfer

Required: No (default is unmanaged)

Description: Sets how the scheduler will handle the preparation overhead of leases. Valid values are:

- unmanaged: The scheduler can assume that there is no deployment overhead, or that some other entity is taking care of it (e.g., one of the enactment backends)
- imagetransfer: A disk image has to be transferred from a repository node before the lease can start.

B.1.5 Option `lease-failure-handling`

Valid values: `cancel`, `exit`, `exit-raise`

Required: No (default is `cancel`)

Description: Sets how the scheduler will handle a failure in a lease. Valid values are:

- cancel: The lease is cancelled and marked as "FAILED"
- exit: Haizea will exit cleanly, printing relevant debugging information to its log.
- exit-raise: Haizea will exit by raising an exception. This is useful for debugging, as IDEs will recognize this as an exception and will facilitate debugging it.

B.1.6 Option `persistence-file`

Valid values: Any string

Required: No (default is `/.haizea/leases.dat`)

Description: This is the file where lease information, along with some additional scheduling information, is persisted to. If set to "none", no information will be persisted to disk, and Haizea will run entirely in-memory (this is advisable when running in simulation, as persisting to disk adds considerable overhead, compared to running in-memory).

B.2 Section `[scheduling]`

The options in this section control how Haizea schedules leases.
This section is required

B.2.1 Option `mapper`

Valid values: Any string

Required: No (default is `greedy`)

Description: VM-to-physical node mapping algorithm used by Haizea. There is currently only one mapper available (the greedy mapper).

B.2.2 Option `policy-admission`

Valid values: Any string

Required: No (default is `accept-all`)

Description: Lease admission policy. This controls what leases are accepted by Haizea. Take into account that this decision takes place before Haizea even attempts to schedule the lease (so, you can think of lease admission as "eligibility to be scheduled").

There are two built-in policies:

- accept-all: Accept all leases.
- no-ARs: Accept all leases except advance reservations.

See the Haizea documentation for details on how to write your own policies.

B.2.3 Option `policy-preemption`

Valid values: Any string

Required: No (default is `no-preemption`)

Description: Lease preemption policy. Determines what leases can be preempted. There are two built-in policies:

- no-preemption: Do not allow any preemptions
- ar-preempts-everything: Allow all ARs to preempt other leases.

See the Haizea documentation for details on how to write your own policies.

B.2.4 Option `policy-host-selection`

Valid values: Any string

Required: No (default is `greedy`)

Description: Physical host selection policy. controls how Haizea chooses what physical hosts to map VMs to. This option is closely related to the mapper options (if the greedy mapper is used, then the greedy host selection policy should be used, or unexpected results will happen).

The two built-in policies are:

- no-policy: Choose nodes arbitrarily
- greedy: Apply a greedy policy that tries to minimize the number of preemptions.

See the Haizea documentation for details on how to write your own policies.

B.2.5 Option `wakeup-interval`

Valid values: A duration in the format `HH:MM:SS`

Required: No (default is `00:01:00.00`)

Description: Interval at which Haizea will wake up to manage resources and process pending requests. This option is not used when using a simulated clock, since the clock will skip directly to the time where an event is happening.

B.2.6 Option `backfilling`

Valid values: `off`, `aggressive`, `conservative`, `intermediate`

Required: No

Description: Backfilling algorithm to use. Valid values are:

- off: don't do backfilling
- aggressive: at most 1 reservation in the future
- conservative: unlimited reservations in the future
- intermediate: N reservations in the future (N is specified in the backfilling-reservations option)

B.2.7 Option `backfilling-reservations`

Valid values: An integer number

Required: Only if

- Option `backfilling` (in section `[scheduling]`) is set to `intermediate`

Description: Number of future reservations to allow when using the "intermediate" backfilling option.

B.2.8 Option `suspension`

Valid values: `none`, `serial-only`, `all`

Required: Yes

Description: Specifies what can be suspended. Valid values are:

- none: suspension is never allowed
- serial-only: only 1-node leases can be suspended
- all: any lease can be suspended

B.2.9 Option `suspend-rate`

Valid values: A real number

Required: Yes

Description: Rate at which VMs are assumed to suspend (in MB of memory per second)

B.2.10 Option `resume-rate`

Valid values: A real number

Required: Yes

Description: Rate at which VMs are assumed to resume (in MB of memory per second)

B.2.11 Option `suspendresume-exclusion`

Valid values: `local`, `global`

Required: No (default is `local`)

Description: When suspending or resuming a VM, the VM's memory is dumped to a file on disk. To correctly estimate the time required to suspend a lease with multiple VMs, Haizea makes sure that no two suspensions/resumptions happen at the same time (e.g., if eight memory files were being saved at the same time to disk, the disk's performance would be reduced in a way that is not as easy to estimate as if only one file were being saved at a time).

Depending on whether the files are being saved to/read from a global or local filesystem, this exclusion can be either global or local.

B.2.12 Option `scheduling-threshold-factor`

Valid values: An integer number

Required: No (default is `1`)

Description: To avoid thrashing, Haizea will not schedule a lease unless all overheads can be correctly scheduled (which includes image transfers, suspensions, etc.). However, this can still result in situations where a lease is prepared, and then immediately suspended because of a blocking lease in the future. The scheduling threshold factor can be used to specify that a lease must not be scheduled unless it is guaranteed to run for a minimum amount of time (the rationale behind this is that you ideally don't want leases to be scheduled if they're not going to be active for at least as much time as was spent in overheads).

The default value is 1, meaning that the lease will be active for at least as much time T as was spent on overheads (e.g., if preparing the lease requires 60 seconds, and we know that it will have to be suspended, requiring 30 seconds, Haizea won't schedule the lease unless it can run for at least 90 minutes). In other words, a scheduling factor of F required a minimum duration of F*T. A value of 0 could lead to thrashing, since Haizea could end up with situations where a lease starts and immediately gets suspended.

B.2.13 Option `override-suspend-time`

Valid values: An integer number

Required: No

Description: Overrides the time it takes to suspend a VM to a fixed value (i.e., not computed based on amount of memory, enactment overhead, etc.)

B.2.14 Option `override-resume-time`

Valid values: An integer number

Required: No

Description: Overrides the time it takes to suspend a VM to a fixed value (i.e., not computed based on amount of memory, enactment overhead, etc.)

B.2.15 Option `force-scheduling-threshold`

Valid values: A duration in the format `HH:MM:SS`

Required: No

Description: This option can be used to force a specific scheduling threshold time to be used, instead of calculating one based on overheads.

B.2.16 Option `migration`

Valid values: `no`, `yes`, `yes-notransfer`

Required: No (default is `no`)

Description: Specifies whether leases can be migrated from one physical node to another. Valid values are:

- no
- yes
- yes-notransfer: migration is performed without transferring any files.

B.2.17 Option `non-schedulable-interval`

Valid values: A duration in the format `HH:MM:SS`

Required: No (default is `00:00:10.00`)

Description: The minimum amount of time that must pass between when a request is scheduled to when it can actually start. The default should be good for most configurations, but may need to be increased if you're dealing with exceptionally high loads.

B.2.18 Option `shutdown-time`

Valid values: A duration in the format `HH:MM:SS`

Required: No

Description: The amount of time that will be allocated for a VM to shutdown. When running in OpenNebula mode, it is advisable to set this to a few seconds, so no operation gets scheduled right when a VM is shutting down. The most common scenario is that a VM will start resuming right when another VM shuts down. However, since both these activities involve I/O, it can delay the resume operation and affect Haizea's estimation of how long the resume will take.

B.2.19 Option `enactment-overhead`

Valid values: A duration in the format `HH:MM:SS`

Required: No

Description: The amount of time that is required to send an enactment command. This value will affect suspend/resume estimations and, in OpenNebula mode, will force a pause of this much time between suspend/resume enactment commands. When suspending/resuming many VMs at the same time (which is likely to happen if suspendresume-exclusion is set to "local"), it will take OpenNebula 1-2 seconds to process each command (this is a small amount of time, but if 32 VMs are being suspended at the same time, on in each physical node, this time can compound up to 32-64 seconds, which has to be taken into account when estimating when to start a suspend operation that must be completed before another lease starts).

B.3 Section `[simulation]`

This section is used to specify options when Haizea runs in simulation
This section is required when:

- Option `mode` (in section `[general]`) is set to `simulated`

B.3.1 Option `clock`

Valid values: `real`, `simulated`

Required: No (default is `real`)

Description: Type of clock to use in simulation:

- simulated: A simulated clock that fastforwards through time. Can only use the tracefile request frontend
- real: A real clock is used, but simulated resources and enactment actions are used. Can only use the RPC request frontend.

B.3.2 Option `starttime`

Valid values: An ISO timestamp: i.e., `YYYY-MM-DD HH:MM:SS`

Required: Only if

- Option `clock` (in section `[simulation]`) is set to `simulated`

Description: Time at which simulated clock will start.

B.3.3 Option `resources`

Valid values: Any string

Required: Yes

Description: Simulated resources. This option can take two values, "in-tracefile" (which means that the description of the simulated site is in the tracefile) or a string specifying a site with homogeneous resources. The format is:

<numnodes> [<resource_type>:<resource_quantity>]+

For example, "4 CPU:100 Memory:1024" describes a site with four nodes, each with one CPU and 1024 MB of memory.

B.3.4 Option `imagetransfer-bandwidth`

Valid values: An integer number

Required: Yes

Description: Bandwidth (in Mbps) available for image transfers. This would correspond to the outbound network bandwidth of the node where the images are stored.

B.3.5 Option `stop-when`

Valid values: `all-leases-done`, `besteffort-submitted`, `besteffort-done`

Required: No (default is `all-leases-done`)

Description: When using the simulated clock, this specifies when the simulation must end. Valid options are:

- all-leases-done: All requested leases have been completed and there are no queued/pending requests.
- besteffort-submitted: When all best-effort leases have been submitted.
- besteffort-done: When all best-effort leases have been completed.

B.3.6 Option `status-message-interval`

Valid values: An integer number

Required: No

Description: If specified, the simulated clock will print a status message with some basic statistics. This is useful to keep track of long simulations. The interval is specified in minutes.

B.4 Section `[accounting]`

Haizea can collect information while running, and save that information to a file for off-line processing. This section includes options controlling this feature.

This section is required

B.4.1 Option `datafile`

Valid values: Any string

Required: No

Description: This is the file where statistics on the scheduler's run will be saved to (waiting time of leases, utilization data, etc.). If omitted, no data will be saved.

B.4.2 Option `probes`

Valid values: Any string

Required: No

Description: Accounting probes.

There are four built-in probes:

- AR: Collects information on AR leases.
- best-effort: Collects information on best effort leases.
- immediate: Collects information immediate leases.
- utilization: Collects information on resource utilization.

See the Haizea documentation for details on how to write your own accounting probes.

B.4.3 Option `attributes`

Valid values: Any string

Required: No

Description: This option is used internally by Haizea when using multiconfiguration files. See the multiconfiguration documentation for more details.

B.5 Section `[deploy-imagetransfer]`

When lease deployment with disk image transfers is selected, this section is used to control image deployment parameters.

This section is required when:

- Option `lease-deployment` (in section `[general]`) is set to `imagetransfer`

B.5.1 Option `transfer-mechanism`

Valid values: `unicast`, `multicast`

Required: Yes

Description: Specifies how disk images are transferred. Valid values are:

- unicast: A disk image can be transferred to just one node at a time
- multicast: A disk image can be multicast to multiple nodes at the same time.

B.5.2 Option `avoid-redundant-transfers`

Valid values: `True` or `False`

Required: No (default is `True`)

Description: Specifies whether the scheduler should take steps to detect and avoid redundant transfers (e.g., if two leases are scheduled on the same node, and they both require the same disk image, don't transfer the image twice; allow one to "piggyback" on the other). There is generally no reason to set this option to False.

B.5.3 Option `force-imagetransfer-time`

Valid values: A duration in the format `HH:MM:SS`

Required: No

Description: Forces the image transfer time to a specific amount. This options is intended for testing purposes.

B.5.4 Option `diskimage-reuse`

Valid values: `none`, `image-caches`

Required: No (default is `none`)

Description: Specifies whether disk image caches should be created on the nodes, so the scheduler can reduce the number of transfers by reusing images. Valid values are:

- none: No image reuse
- image-caches: Use image caching algorithm described in Haizea publications

B.5.5 Option `diskimage-cache-size`

Valid values: An integer number

Required: Only if

- Option `diskimage-reuse` (in section `[deploy-imagetransfer]`) is set to `True`

Description: Specifies the size (in MB) of the disk image cache on each physical node.

B.6 Section `[tracefile]`

When reading in requests from a tracefile, this section is used to specify the tracefile and other parameters.

This section is optional.

B.6.1 Option `tracefile`

Valid values: Any string

Required: Yes

Description: Path to tracefile to use.

B.6.2 Option `imagefile`

Valid values: Any string

Required: No

Description: Path to list of images to append to lease requests. If omitted, the images in the tracefile are used.

B.6.3 Option `injectionfile`

Valid values: Any string

Required: No

Description: Path to file with leases to "inject" into the tracefile.

B.6.4 Option `runtime-slowdown-overhead`

Valid values: A real number

Required: No

Description: Adds a runtime overhead (in %) to the lease duration.

B.6.5 Option `add-overhead`

Valid values: `none`, `all`, `besteffort`

Required: No (default is `none`)

Description: Specifies what leases will have a runtime overhead added:

- none: No runtime overhead must be added.
- besteffort: Add only to best-effort leases
- all: Add runtime overhead to all leases

B.6.6 Option `bootshutdown-overhead`

Valid values: A duration in the format `HH:MM:SS`

Required: No

Description: Specifies how many seconds will be alloted to boot and shutdown of the lease.

B.6.7 Option `override-memory`

Valid values: An integer number

Required: No (default is `-1`)

Description: Overrides memory requirements specified in tracefile.

B.7 Section `[opennebula]`

This section is used to specify OpenNebula parameters, necessary when using Haizea as an OpenNebula scheduling backend.

This section is required when:

- Option `mode` (in section `[general]`) is set to `opennebula`

B.7.1 Option `host`

Valid values: Any string

Required: Yes

Description: Host where OpenNebula is running. Typically, OpenNebula and Haizea will be installed on the same host, so the following option should be set to 'localhost'. If they're on different hosts, make sure you modify this option accordingly.

B.7.2 Option `port`

Valid values: An integer number

Required: No (default is `2633`)

Description: TCP port of OpenNebula's XML RPC server

B.7.3 Option `stop-when-no-more-leases`

Valid values: `True` or `False`

Required: No

Description: This option is useful for testing and running experiments. If set to True, Haizea will stop when there are no more leases to process (which allows you to tun Haizea and OpenNebula unattended, and count on it stopping when there are no more leases to process). For now, this only makes sense if you're seeding Haizea with requests from the start (otherwise, it will start and immediately stop).

B.7.4 Option `dry-run`

Valid values: `True` or `False`

Required: No

Description: This option is useful for testing. If set to True, Haizea will fast-forward through time (note that this is different that using the simulated clock, which has to be used with a tracefile; with an Haizea/OpenNebula dry run, you will have to seed OpenNebula with requests before starting Haizea). You will generally want to set stop-when-no-more-leases when doing a dry-run.

IMPORTANT: Haizea will still send out enactment commands to OpenNebula. Make sure you replace onevm with a dummy command that does nothing (or that reacts in some way you want to test; e.g., by emulating a deployment failure, etc.)

C XML format reference

Haizea uses XML to encode certain information, most notably in the LWF (Lease Workload File) files. This chapter describes three XML elements that are used in various Haizea components, and also describes the LWF format.

C.1 Nodes element

A `<nodes>` element is used to describe a collection of machines ("nodes"), such as those required by a lease or those in a simulated site. Instead of describing each node individually, the `<nodes>` stores information on each distinct node capacity along with the number of nodes with that capacity. For example, if a lease required 5 nodes with 1024MB of memory and 10 nodes with 2048GB of memory, the `<nodes>` will have two "sets of nodes", instead of 15 individual entries.

The root `<nodes>` element has no attributes:

```
<nodes>
  ...
</nodes>
```

And must contain one or more `node-set` element:

```
<node-set numnodes="...">
  ...
</node-set>
```

Each `<node-set>` element represents a "set of nodes". The `numnodes` attribute is used to denote the number of nodes with the same capacity. This capacity is described with one or more `<res>` elements:

```
<res type="..." amount="..."/>
```

The `type` attribute specifies the type of resource (`CPU`, `Memory`, etc.), and `amount` specifies the amount of that resource in each node (the amount must be a positive integer). There resource type must be a string and, although the `<res>` element places no restrictions on this string (i.e., you can use any arbitrary resource you want), the `<lease>` and `<site>` elements (described below) do place some restrictions on it.

C.1.1 Example

The following specifies a collection of 12 nodes, all with one CPU, four with 1024MB of memory and eight with 2048MB of memory.

```
<nodes>
  <node-set numnodes="4">
    <res type="CPU" amount="100"/>
    <res type="Memory" amount="1024"/>
  </node-set>
  <node-set numnodes="8">
    <res type="CPU" amount="100"/>
    <res type="Memory" amount="2048"/>
  </node-set>
</nodes>
```

C.2 Lease element

The `<lease>` element is used by Haizea's XML-RPC API to send lease requests to Haizea, and to return lease information to the client. It is also used in the LWF format to describe lease requests.

The `<lease>` element has two attributes: `id`, a unique integer identifier (assigned by Haizea), and `preemptible`, indicating whether the lease can be safely preempted (`yes` or `no`)

```
<lease id="..." preemptible="...">
  ...
</lease>
```

The `<lease>` element has four child elements:

- `<nodes>` (as described above): Specifies the nodes requested by the lease, and the capacity required in these nodes.
- `<duration>`: The requested duration of the lease. This element has a single attribute `time` which is used to specify a duration with format `DD:HH:MM.SS` (DD: days; HH: hours; MM: minutes; SS: seconds)
- `<start>`: The requested started time of the lease. If the element is empty, this means the requested starting time is unspecified, and it is up to Haizea to determine the starting time on a best-effort basis:

  ```
  <start/>
  ```

  If an exact starting time is specified, then this element has an `<exact>` child element with a `time` attribute specifying the starting time using an ISO timestamp (`YYYY-MM-DD HH:MM:SS`) or a relative starting time (`+DD:HH:MM:SS`), which is interpreted as being relative to the time the lease is submitted to Haizea.

  ```
  <start>
    <exact time="..."/>
  </exact>
  ```

If the lease is requested to start immediately, then this element has an empty `<now>` child element:

```
<start>
  <now/>
</exact>
```

- `<software>`: The software environment required by the lease. Currently, Haizea only supports specifying a lease's software environment as being contained inside a disk image:

```
<software>
  <disk-image id="..." size="..."/>
</software>
```

The `disk-image` child element specifies information about the disk image: `id`, a unique identifier, and `size`, its size in megabytes.

Additionally, the software environment can be left unspecified, which means that Haizea can assume that setting up the software environment will be handled by another entity:

```
<software>
  <none/>
</software>
```

C.2.1 Example

The following specifies a best-effort lease, requesting one node with one CPU and 1024 MB of memory, for one hour, and a software environment contained in diskimage `foobar1.img` (a 1GB image).

```
<lease id="1" preemptible="true">
  <nodes>
    <node-set numnodes="1">
      <res amount="100" type="CPU"/>
      <res amount="1024" type="Memory"/>
    </node-set>
  </nodes>
  <start/>
  <duration time="01:00:00"/>
  <software>
    <disk-image id="foobar1.img" size="1024"/>
  </software>
</lease>
```

C.3 Site element

The `<site>` element is used in LWF files to describe the site the lease workload is meant to be run on. In future versions of Haizea, it will also be used to specify simulated sites from the configuration file.

The `<site>` has two child elements: `<resource-types>`, used to specify the valid resource types in the site (separated by spaces), and a `<nodes>` element specifying the nodes in the site:

```
<site>
  <resource-types names="..."/>
  <nodes>
    ...
  </nodes>
</site>
```

The nodes can only have resources specified in `<resource-types>`. For example, if `"CPU Memory"` is specified, then the `type` attribute of the `<res>` elements in `<nodes>` can only be `CPU` or `Memory`. Note that Haizea doesn't "understand" what these types are, and treats them all as consumable capacities, so you can specify any resource types you want. However, Haizea does require that, at the very least, all sites specify at least a `CPU` and a `Memory` resource.

C.3.1 Example

The following specifies a site supporting three types of resources: `CPU`, `Memory`, and `Ponies`. Again, Haizea does not interpret the resource type names (other than `CPU` and `Memory`), so it is up to you to interpret what the `Ponies` resource type means, and what it means for a lease to request ponies.

The site has eight nodes, all with one CPU and 1024MB of memory. Four of them have four ponies, and the other four have none (if no amount is specified for a resource type, it defaults to zero).

```
<site>
  <resource-types names="CPU Memory Ponies"/>
  <nodes>
    <node-set numnodes="4">
      <res type="CPU" amount="100"/>
      <res type="Memory" amount="1024"/>
      <res type="Ponies" amount="4"/>
    </node-set>
    <node-set numnodes="4">
      <res type="CPU" amount="100"/>
      <res type="Memory" amount="1024"/>
    </node-set>
  </nodes>
</site>
```

C.4 LWF file format

The LWF (Lease Workload Format) describes a workload of leases that can be used by Haizea when running in simulation mode. In this workload, the starting time is `00:00:00:00`, and all

times are specified relative to that starting time. For example, an AR lease requested to start at `00:01:00:00` starts one hour after the start of the workload. Each lease also has an `arrival time`, the time at which the lease is submitted to Haizea.

The root element of an LWF file is the `<lease-workload>` element:

```
<lease-workload name="...">
  <description>
  ...
  </description>

  <site>
    ...
  </site>

  <lease-requests>
    ...
  </lease-requests>
</lease-workload>
```

This element has a single attribute `name`, with the name of this workload. The `<description>` child element can be used to provide a longer description of the workload. The `<site>` element is used to specify the site the workload is meant to be run on, and the `<lease-request>` element contains the actual lease requests. Each `<lease>` element is wrapped inside a `<lease-request>` element:

```
<lease-request arrival="...">
  <realduration time="..."/>
  <lease ...>
      ...
  </lease>
</lease-request>
```

This element has an attribute `arrival` indicating when the lease is submitted to Haizea. It can also have an *optional* `<realduration>` child element specifying the real duration of the lease. In many systems, users request resources for a period of time, but relinquish the resources earlier. When simulating workloads, it is important to take this information into account, since the simulator must stop the lease at the end of that "real duration", not at the end of the full requested duration. Note that, if a `<realduration>`, Haizea will still schedule the lease assuming it is going to use its full requested duration (since, like a non-simulated scheduler, it can't assume to have a priori knowledge of when the lease will really end), but the simulator will generate an event indicating the lease has ended prematurely when its "real duration" has elapsed.

D Accounting probes reference

D.1 ARProbe

Full name: `haizea.pluggable.accounting.leases.ARProbe`
Short name: `ar`
Description:
Collects information from Advance Reservation leases

- Counters
  - "Accepted AR": Number of accepted AR leases
  - "Rejected AR": Number of rejected AR leases
- Per-run data
  - "Total accepted AR": Final number of accepted AR leases
  - "Total rejected AR": Final number of rejected AR leases

D.2 BEProbe

Full name: `haizea.pluggable.accounting.leases.BEProbe`
Short name: `best-effort`
Description:
Collects information from best-effort leases

- Counters
  - "Best-effort completed": Number of best-effort leases completed throughout the run
  - "Queue size": Size of the queue throughout the run
- Per-lease data
  - "Waiting time": Time (in seconds) the lease waited in the queue before resources were allocated to it.
  - "Slowdown": Slowdown of the lease (time required to run the lease to completion divided by the time it would have required on a dedicated system)
- Per-run data
  - "Total best-effort completed": Final number of completed best-effort leases
  - "all-best-effort": The time (in seconds) when the last best-effort lease was completed.

D.3 IMProbe

Full name: haizea.pluggable.accounting.leases.IMProbe
Short name: immediate
Description:
Collects information from immediate leases

- Counters
  - "Accepted Immediate": Number of accepted Immediate leases
  - "Rejected Immediate": Number of rejected Immediate leases
- Per-run data
  - "Total accepted Immediate": Final number of accepted Immediate leases
  - "Total rejected Immediate": Final number of rejected Immediate leases

D.4 CPUUtilizationProbe

Full name: haizea.pluggable.accounting.utilization.CPUUtilizationProbe
Short name: cpu-utilization
Description:
Collects information on CPU utilization

- Counters
  - "CPU utilization": Amount of CPU resources used in the entire site at a given time. The value ranges between 0 and 1.

D.5 DiskUsageProbe

Full name: haizea.pluggable.accounting.utilization.DiskUsageProbe
Short name: disk-usage
Description:
Collects information on disk usage

- Counters
  - "Disk usage": Maximum disk space used across nodes.